2012

Device Dash: Designing, Implementing, and Evaluating an Educational Computer Security Game

Era Vuksani
Wellesley College, evuksani@wellesley.edu

WELLESLEY COLLEGE & MIT LINCOLN LABORATORY

Device Dash: Designing, Implementing, and Evaluating an Educational Computer Security Game

by

Era Vuksani

Submitted in Partial Fulfillment of
the Prerequisite for Honors

in the
COMPUTER SCIENCE DEPARTMENT

May 2012

"Play it - before you live it."

Stefanie Olsen

WELLESLEY COLLEGE & MIT LINCOLN LABORATORY

Abstract

Era Vuksani

Advisors: Tyler MOORE (Wellesley College), Richard LIPPMANN (MIT Lincoln Laboratory), and Tamara YU (MIT Lincoln Laboratory)

This thesis describes the making and testing of an educational game, Device Dash, that is designed to teach important computer security concepts normally described in technical documents, such as the SANS Critical Controls, in a fun and more easily understandable way. We discuss the different aspects of designing an educational game, as well as our own vision for our game. A detailed level-by-level overview of Device Dash is provided, in which we explain its teaching goals and its gameplay. To evaluate Device Dash's effectiveness for teaching computer security, we conducted a series of tests to measure the players' knowledge before and after playing the game. The results suggest that Device Dash improves the players' knowledge about the security risks in a network and methods to reduce these risks.

Acknowledgements

With the most heartfelt appreciation, I thank my advisors, Tyler Moore, Richard Lippmann, and Tamara Yu, for their constant guidance and involvement in this paper and in the creation of Device Dash. Without them, this thesis would not have been possible. Also, many thanks to my committee for providing feedback and taking part in this process with me.

Thank you also to my aunt, who provided me with a great deal of guidance and helped me understand LaTeX.

Last, but not least, I would like to thank my friends for being there whenever I needed a second opinion, testers for my game, or people to discuss my thesis with. (Special thanks to Diana Tantillo and Ginger Chou for their support.)

Contents

List of Figures

List of Tables

Dedicated to my parents and my aunt for believing in me when I didn't believe in myself.

Chapter 1

Introduction

"Just play. Have fun. Enjoy the game."

Michael Jordan

1.1 Computer Security

Computer security is critical in today's world. In recent years, there have been countless instances of security breaches. One highly publicized example is the breach into TJX's network, resulting in 45.6M stolen card numbers [15]. In 2012 alone, Verizon had 855 incidents, resulting in 174 million records compromised overall [1]. The need for better security is evident. Organizations, such as the SANS Institute, have devised key recommendations for defense implementations. The SANS' 20 Critical Security Controls [13] explain the steps that should be undertaken to thwart attacks on corporate networks.

A study done by the Verizon RISK Team in cooperation with the Australian Federal Police, the Dutch National High Tech Crime Unit, the Irish Reporting & Information Security Service, the Police Central e-Crime Unit (UK), and the United States Secret Service [1] explains that, for 63% of breaches in all organizations, the costs of the recommended preventive measures are simple and cheap (see Figure 1.1). This implies that, even though there are controls available to counter these attacks, organizations are not using them. People are not responding to conventional ways to increase awareness. Educational games can be used as an alternate way to increase computer security education in a fun way.

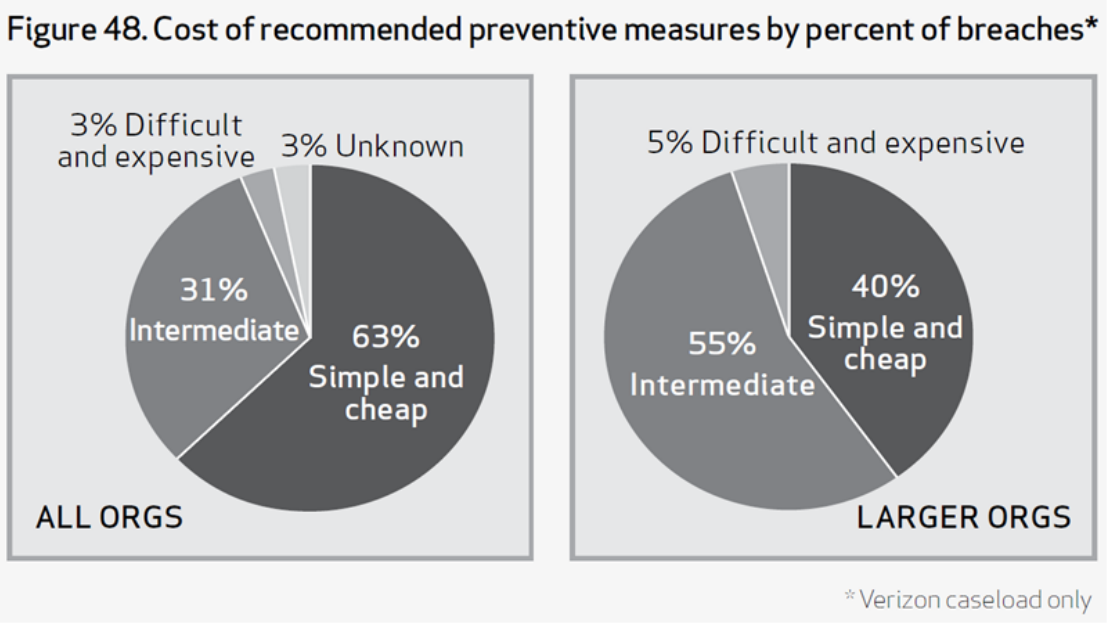


FIGURE 1.1: Verizon's Investigation on Security Breaches

1.2 Educational Games

Many of us growing up in the digital age, at one point or another, have been criticized for spending too much time playing video games. Perceived as "mindless entertainment" and likened to television, which rots young brains, video games have often acquired a negative reputation. However, not all video games are bad. Some video games incorporate both learning and entertainment to teach a variety of useful concepts. Studies suggest that the best way to learn is through active learning, involving participation, discussions (internal or external) and critical thinking [14]. Educational video games seem to be an ideal medium to teach in this way.

1.3 Why Play Games At All?

What is it that draws us to playing? Why do we sigh and groan when we have to study from textbooks, but become excited at the thought of playing a video game? It is not because video games do not make us think and learn; it is because video games provide entertainment.

By offering the players an escape from their routine-based life, games entrap the players in their wondrous fictional worlds, where anything is possible, or, at least, so it may seem. It is fun to learn the rules of a new universe, try to break those rules (or break

the game, which can also be quite fun), discover new possibilities, and become fully immersed in that universe. The game's job is to grab the players' attention quickly and to ensure that they would rather play the game than do anything else.

Players may become so captivated by all of the "pretty" things that the game has to offer at each new level, such as clothing, weapons and cool skill effects in Massively Multiplayer Online Role-Playing Games (MMORPGs), that this alone could keep them playing level after level. Or they may become so wrapped up in the story line, in its puzzles (such as the physics and portal creations in the *Portal* games), or in its communities (such as the factions or guilds in *Perfect World International*), that they would want to keep exploring the game. Players also want to have the satisfaction of beating the game; they want to be able to boast that they could solve it without any hints.

Some gamers play because they want to learn something, from a technique for winning *Solitaire* to learning about history in *Where In The World Is Carmen Sandiego.* Video games have a great potential: they have the ability to intrinsically teach without the players' realizing it. When done right, games will captivate the players, so that they will not want to stop playing and learning.

This was exactly what we wanted to do. At MIT Lincoln Laboratory, we wanted to create a game that would teach the players useful computer security concepts, based on the 20 Critical Controls published by SANS [13] without relinquishing fun, the game's overall appeal to all ages, and its immersive ability. We called this game Device Dash, after *Diner Dash* (see Chapter 3).

1.4 Device Dash

The purpose of Device Dash is to teach anyone about the security risks in a network, how to minimize those risks, and how to mitigate potential attacks. Device Dash features concepts such as authorized and unauthorized devices, administrative power and privileges, and boundary defenses. The game starts out slowly, to get the players acquainted with the network. As the players progress through the levels, there is an increase in difficulty and speed, and the players have to juggle a multitude of tasks, such as authorizing users to be on the network, removing unauthorized users that are on the network, removing compromised devices from the network, etc. Through experimentation, the player learns strategies (or Critical Controls) that would help him manage the network more effectively.

Since Device Dash is based on 3 of the 20 Critical Controls, in Chapter 2, we begin by giving an overview of the 20 Critical Controls, then introducing the three concepts chosen to be taught by our game. After reviewing existing educational security games, in Chapter 3, we explain the purpose and requirements of Device Dash. Chapter 4 describes Device Dash's gameplay in greater detail; we explain the different scenarios, as well as the related choices that the player needs to make, according to rules, and each of their purposes. At the end of each level, a number of messages are available in order to guide the players to making the best choice possible during the upcoming level. At the same time, these messages provide a fun way to teach the player about the security procedures being taught. Moreover, to make the game more attractive and strategic, the players can buy a number of items with their points. In Chapter 5, we briefly describe our code implementation.

We discuss our testing strategy in Chapter 6. We explain our decisions about the skills to test, and we discuss the data that we chose to acquire and analyze for an understanding of the player's evolution through the game process. The game's evaluation is presented in Chapter 7, through a series of figures and discussions. By testing Device Dash, we found out that players did learn from playing the game. However, our testing was inconclusive when it came to comparing the amount of learning from our game to the amount of learning from the control group's non-educational game. Possible reasons for this are discussed at the end of the chapter.

Finally, some concluding remarks, alterations to our testing process, and future work are presented in Chapter 8. In addition, Appendix A contains the introduction that players read for the reading-based testing, Appendix B contains the questions asked for each of the testing phases, Appendix C contains an overview of the game's items, Appendix D contains the complete data for our testing results, and Appendix E contains the code for Device Dash.

Chapter 2

Critical Security Controls

"The price of liberty is eternal vigilance."

Thomas Jefferson

In this chapter, we will present an overview of SANS' 20 Critical Security Controls (Section 2.1), followed by detailed explanations of the three Critical Controls used in our game: Critical Control 1: Inventory of Authorized and Unauthorized Devices (Section 2.2), Critical Control 5: Boundary Defense (Section 2.3), and Critical Control 8: Controlled Use of Administrative Privileges (Section 2.4).

2.1 Background on the Critical Security Controls

Cyber security has become increasingly important over the years. A series of high-profile incidents, such as the breach into TJX Companies Inc., resulting in 45.6 million card numbers being stolen [15], underscores the need to protect networks from similar attacks in the future. One way to achieve this is through the implementation of an automated monitoring defense system, which has the "ability to automatically test and validate whether current security measures are working and proactively remediate vulnerabilities in a timely manner" [13]. In that spirit, the SANS Institute recommends 20 Critical Security Controls (Critical Controls or CCs for short) [13], which, once implemented, would help prepare organizations for today's most critical threats. Most of the 20 Critical Controls are easy to implement, since they focus on tool-based assessments, which can be automated.

The Critical Controls are already being used in government agencies and large enterprises, in the hope that all companies will adopt these controls in the near future. While

the utilization of these controls does not guarantee that computers and data will be kept safe from attackers, the Critical Controls are important steps toward improving computer security in organizations.

This work will focus on version 3.0 of the Critical Controls and, particularly, on sections of three of these controls:

- Critical Control 1: Inventory of Authorized and Unauthorized Devices
- Critical Control 5: Boundary Defense
- Critical Control 8: Controlled Use of Administrative Privileges

Our game, Device Dash will familiarize the player with these controls, which we explain in detail in the rest of this chapter.

2.2 Critical Control 1: Inventory of Authorized and Unauthorized Devices

Critical Control 1 addresses a type of attack where attackers scan the address spaces of organizations, looking for unauthorized devices that have not been checked for malware and that tend to be the most vulnerable due to their potential lack of installed security updates or patches. These scans are often done automatically and continuously. Once attackers gain control over a device, they can use that device as an entry point to break into other devices on the same network. These devices might not have been previously penetrable from outside of the network. Therefore, it is necessary to have an automated and continuous mechanism to detect such attacker scans or backdoors.

In the real world, unauthorized devices are often detected by administrators (or admins) using an asset inventory, containing the information (MAC address, IP address, device type, device name, owner, etc.) of all authorized devices. Furthermore, each device's traffic may be analyzed, and the devices may be continuously monitored for unusual or suspicious behavior. (This could also be done through continuously monitoring the system by hand, but is highly inefficient.)

Network Access Control (NAC) can also be implemented. NAC allows only devices that have met the necessary requirements, including being up to date with security patches and having up-to-date antivirus software, to connect to the company's network.

2.3 Critical Control 5: Boundary Defense

Attackers can use a single compromised device, especially one belonging to a network administrator, as a pivoting point to get access to other devices on the same network. By breaking into this device, attackers can use their location within the network to attack other company computers. It is often easier to attack from within a network than it is to attack from the outside, due to fewer inter-networking restrictions. Consequently, there is a potential for the spread of infections to many devices, compromising all of their data and resulting in a global attack on a flat network[1].

To thwart this, system admins should use boundary defenses. For instance, they could subdivide the network into multiple subnetworks (subnets). This would contain any infections that occur within a subnet and keep them from spreading outside of that subnet, if each boundary is properly secured. This would also make cleaning up the subnet after such an attack easier and more manageable.

2.4 Critical Control 8: Controlled Use of Administrative Privileges

Administrative users have abilities and privileges that regular users do not, such as installing software and changing permissions. This means that administrators have to be more careful than other users, since their actions have more consequences. Downloading and opening a questionable attachment, for example, could mean infecting the entire computer system and all of its users, which could result in the attackers' acquiring sensitive data on that system.

To prevent this, administrators and their accounts should be monitored more closely. Automated and continuous defense systems, such as address space scanners or NAC, should be used, and activities on these systems should be carefully audited. Furthermore, admins could implement an alert system in order to notify them of any new unauthorized devices that have joined the system and any suspicious behavior on the network (even from other admins). The alert system should alert the system admins of such a device every 24 hours, until the device has been removed from the network.

These sections from the Critical Controls are further discussed in both Appendix A and in Section 4.3.

[1] A flat network is a network in which all computers are directly connected to one another. [17]

Chapter 3

Game Design

"A voluntary activity or occupation executed within certain fixed limits of time and place, according to rules freely accepted but absolutely binding, having its aim in itself and accompanied by a feeling of tension, joy, and the consciousness that it is different from ordinary life."

Johan Huizinga, Definition of 'play' in "Homo Ludens", 1968

In this chapter, we aim to explain our game design process. Section 3.1 discusses our initial game ideas, goals, and concerns. Section 3.2 reviews related educational games. Section 3.3 presents the style of the game that we wanted to make.

3.1 Background, Purpose, and Requirements of Device Dash

The initial purpose of Device Dash was to teach company employees network security to work more effectively and make better decisions. We wanted to make a game not only because games are fun, but because we believe that games can teach more than other forms of teaching, such as literature, can. Games allow the players to get deeply immersed in the gameplay and to learn, sometimes without even realizing it. In addition, games can portray content in a more engaging way than papers do, as well as allow the players to make their own decisions and see where these choices take them. By adding a visual component to the content being taught, we hope that the player will also be able to retain the material better in the long run.

Creating a game based on the Critical Controls from scratch would be a challenging and labor-intensive task, since it involves explaining each of the 20 CCs in detail and

making sure that the audience understands them and their implications, and doing so in an engaging way. We decided to focus on Critical Control 1: Inventory of Authorized and Unauthorized Devices. However, as time went on and the game matured, all of us became very excited at the prospect of having a game that would be able to teach even more. We decided to add two more CCs to the list: Boundary Defense (CC 5) and Controlled Use of Administrative Privileges (CC 8). These three Critical Controls have been analyzed extensively by MIT Lincoln Laboratory.

With the addition of these two Critical Controls came problems, but also new ideas. We still could not use all of the three CCs because there were parts that were almost, if not completely, impossible to model accurately or that would be too boring to play. Therefore, we decided to pick certain aspects of the three CCs that were important, but that would also be memorable and enjoyable to play with. An example of this are the scanners implemented in Device Dash, which are visually pleasing, since the players are able to see what the scanners are currently scanning.

Furthermore, we decided that, although we had focused on teaching company employees in charge of security decisions the importance of the Critical Controls, we also wanted to tailor Device Dash to anyone who wants to understand these concepts. Thus, we turned Device Dash into an educational game for everyone.

3.2 Some Existing Types of Games

Since we had never made any educational games before, we decided to peruse through some of the educational games in the market, in order to find out what aspects they taught, in what way they portrayed them, and how they taught them to their players. In addition, we also went to the Singapore-MIT GAMBIT Game Lab and discussed educational games and formats. Among all of the games that we looked at, we found three to be particularly important, as well as interesting in their messages and in their teaching styles.

3.2.1 *Anti-Phishing Phil*

Anti-Phishing Phil [9] is an educational game that teaches the signs of potentially unsafe URLs. In *Anti-Phishing Phil*, the player takes on the character of a young fish, Phil, who is learning about which URLs, portrayed as worms, are safe to eat (ie: to click on) and which are not (see Figure 3.1). However, the signs for safe and unsafe URLs vary greatly from URL to URL: what is a potentially safe URL in one case could be a dangerous

URL in another case. This makes it hard to generalize learning from ambiguous signs, which is one of the drawbacks of *Anti-Phishing Phil*. Another drawback is that it has limited content: it only teaches about URLs.

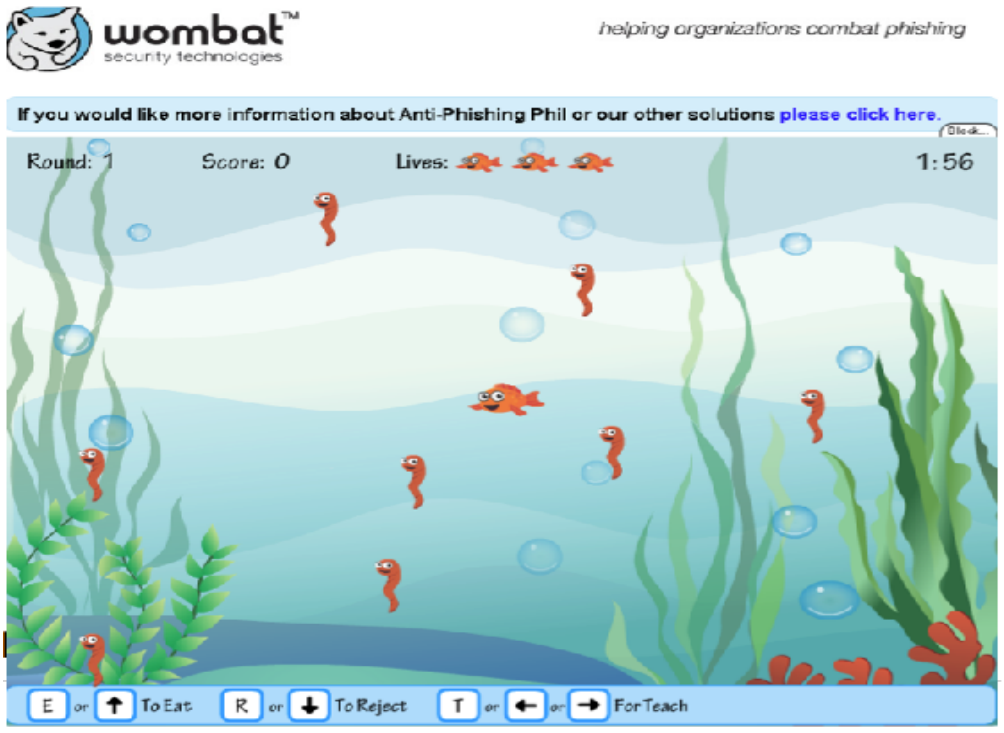


FIGURE 3.1: *Anti-Phishing Phil*

3.2.2 *OnGuard Online*

Second, we looked at *OnGuard Online*, whose purpose is to teach safe behaviors to protect personal and business information and assets. In *OnGuard Online*, the player is given different scenarios. One scenario is that the player is a secret agent and his laptop has been stolen once before. Therefore, the player has to be careful about the rules that he adheres to, in order to make sure that it is not stolen again or that his sensitive information does not fall in the hands of someone else. Figure 3.2 shows *OnGuard Online*'s gameplay.

The player has the choice of which and how many scenarios he wants to play. For each scenario, he is asked a number of questions, out of which he must answer a certain number correctly, in order to safeguard his laptop. Although this game does have a wide range of scenarios and has the potential to teach a great deal because of the complexities that it deals with, it also has many drawbacks. Its scenarios, although interesting and varied, are long and repetitive in structure. The answers to the questions are, very often,

FIGURE 3.2: *OnGuard Online*

easily guessable, not to mention that sometimes there are also multiple correct answers, which makes guessing the right answer more probable.

3.2.3 *CyberCIEGE*

Finally, we reviewed *CyberCIEGE* [12], a complex security game that deals with virtual 3-D scenarios. The purpose of *CyberCIEGE* is to teach computer and network security concepts and safe practices, such as malware containment, ID theft protection, password security, physical security, link encryption, and safe browsing. In *CyberCIEGE*'s main scenario, the player takes on the role of the system administrator and has to implement a range of resources in order to keep his enterprise secure: Figure 3.3 demonstrates some of the different interfaces of *CyberCIEGE*.

The player chooses what to implement based on the needs of the enterprise and its employees. He can see the thoughts of the employees, along with how well he is doing, which is shown using a metric of dollars. Although *CyberCIEGE* has a great concept and the visualizations are both attractive and informative, it has way too many controls, making certain implications hard to understand: for example, the player can guess that allowing passwords on post-its will make the organization less secure, but how so? In

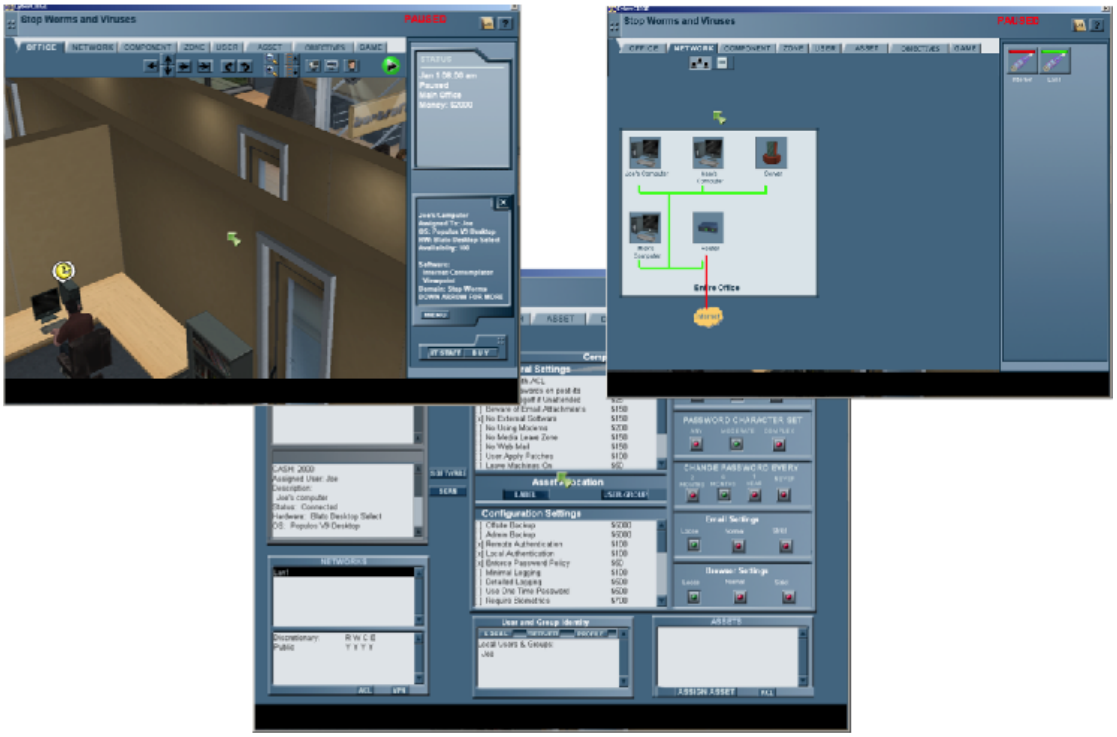


FIGURE 3.3: *CyberCIEGE*

CyberCIEGE, it is hard to discern the cause and effect of actions, until they have already been performed and cannot be undone.

3.3 Choosing Game Play Style

After evaluating these games, we came up with our game's basic idea: a network admin that has to cater to his company's employees. We wanted the game to be challenging, but enjoyable; it should frustrate players to the point that they would want to play it again, without making them want to give up. At the same time, of course, the players should understand and retain the information that the game is presenting better than if they had read about it. In addition, we talked to the Singapore-MIT GAMBIT Game Lab's Philip Tan Boon Yew and a few other staff and faculty about our ideas and how to incorporate them into the game that we wanted to make. They suggested that we look into a list of different games and explained their concepts of to us. A game that seemed to have features similar to what we wanted our game to have was *Diner Dash*, which is shown in Figure 3.4.

In *Diner Dash*, the player takes on the role of a waitress, who runs around the restaurant trying to sit people at tables, take their orders, hand their orders to the chef, take their

FIGURE 3.4: *Diner Dash*

orders back to their table, hand them their checks, and clear their tables. Scoring is given in terms of dollars earned by the waitress, including tips for timely service. *Diner Dash* seemed an appropriate stylistic choice:*Diner Dash* has a waitress that caters to customers, just like Device Dash has a system administrator that caters to employees. In *Diner Dash*, the player has many options to choose from, in terms of what he should do next, just as in Device Dash: the player can, for example, decide to sit a couple down before taking the order of another table or he can remove the dirty dishes from one table, before giving the order of another table to the chef. Hence, we decided to make a game that was similar in form to *Diner Dash*, but that, in content, was very different because it dealt with network security.

Chapter 4

Game Description and Messages

"You have to PLAY the game, to find out WHY you're playing the game."

David Cronenberg

The following chapter describes the game in detail. Section 4.1 recounts the plot of the game, Section 4.2 talks about the basic mechanics of the game, while 4.3 discusses the items, messages, and gameplay for each level in detail. A complete list of the items available in Device Dash, their requirements, and their descriptions can be found in Appendix C.

4.1 Device Dash's Story

In Device Dash, the player assumes the role of the system administrator of a large company. He goes through the stages of receiving his offer letter from the company, his first day at work, both of which are taught during the tutorial, and the succeeding days at work. His job is to review all of the devices (laptop, iPod, etc.) that employees want to connect to the company's network.

The admin/player sits at his desk, in front of which employees with devices to connect to the network form a line. It is the admin's job to allow as many devices as he can to access the company's network. At the same time, he must constantly scan the company's IP address space, checking for unauthorized or compromised devices. Unauthorized devices are devices that the admin has not allowed to be on the network, while compromised devices are devices that are infected with malware. The admin must remove these devices from the network. Intially, all of these tasks must be done manually, which can become quite frustrating, once the size of the address space as well as the speed with

which people come to the desk increases. To repress these problems, the player can opt to buy items that will make his job easier by making these tasks more manageable.

4.2 Basic Game Mechanics

In order to understand the mechanics of the game, the player must read the tutorial first. The tutorial starts out by familiarizing the player with the environment and explaining buttons and shortcuts, as shown in Figures 4.1-4.2.

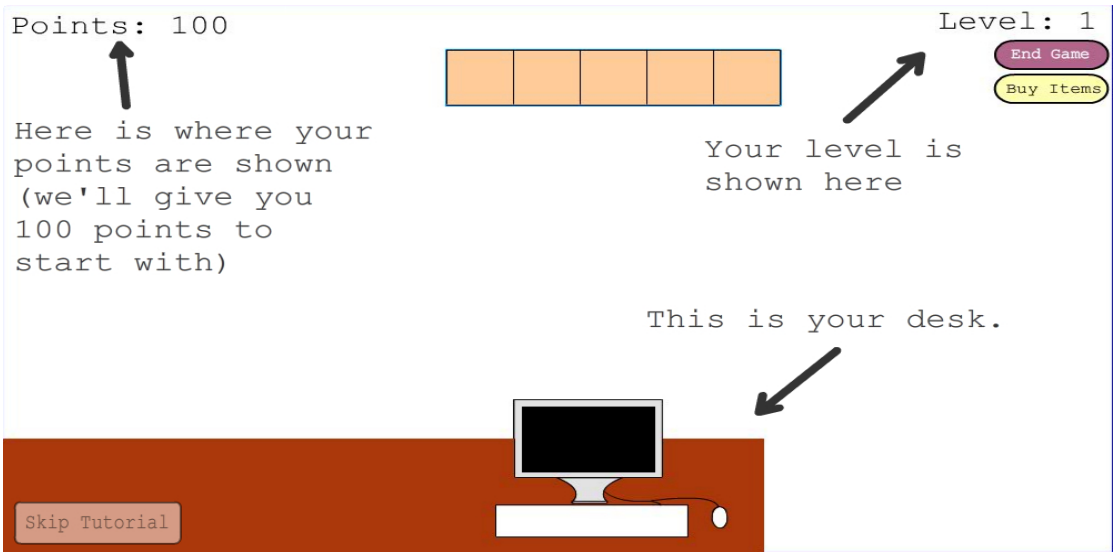


FIGURE 4.1: Tutorial Images - Familiarization with Environment

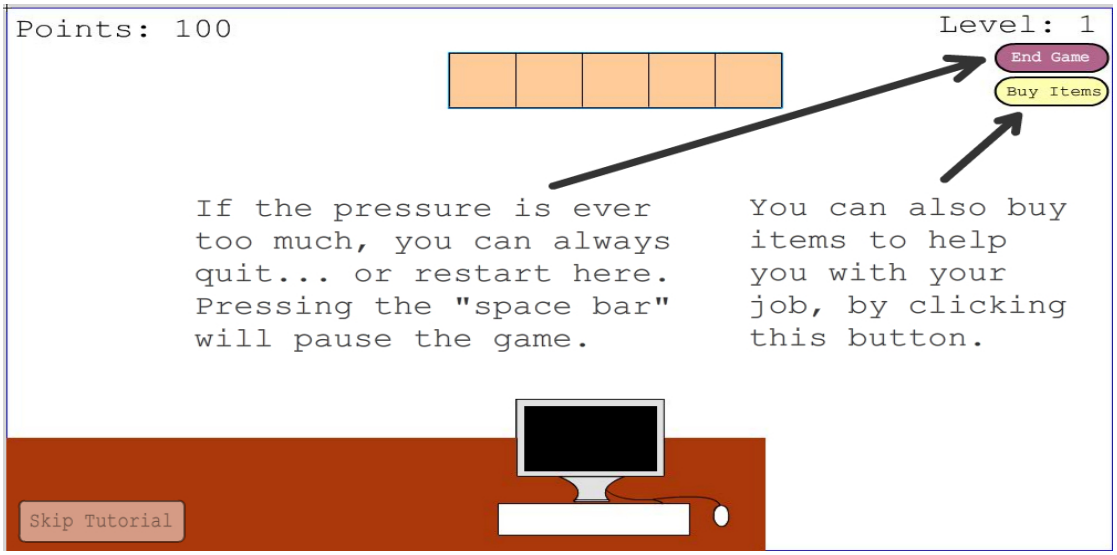


FIGURE 4.2: Tutorial Images - Buttons and Shortcuts

The game is simple to understand. It begins with employees coming down from the top-leftmost corner of the screen and moving to the player's desk (Figure 4.3). There,

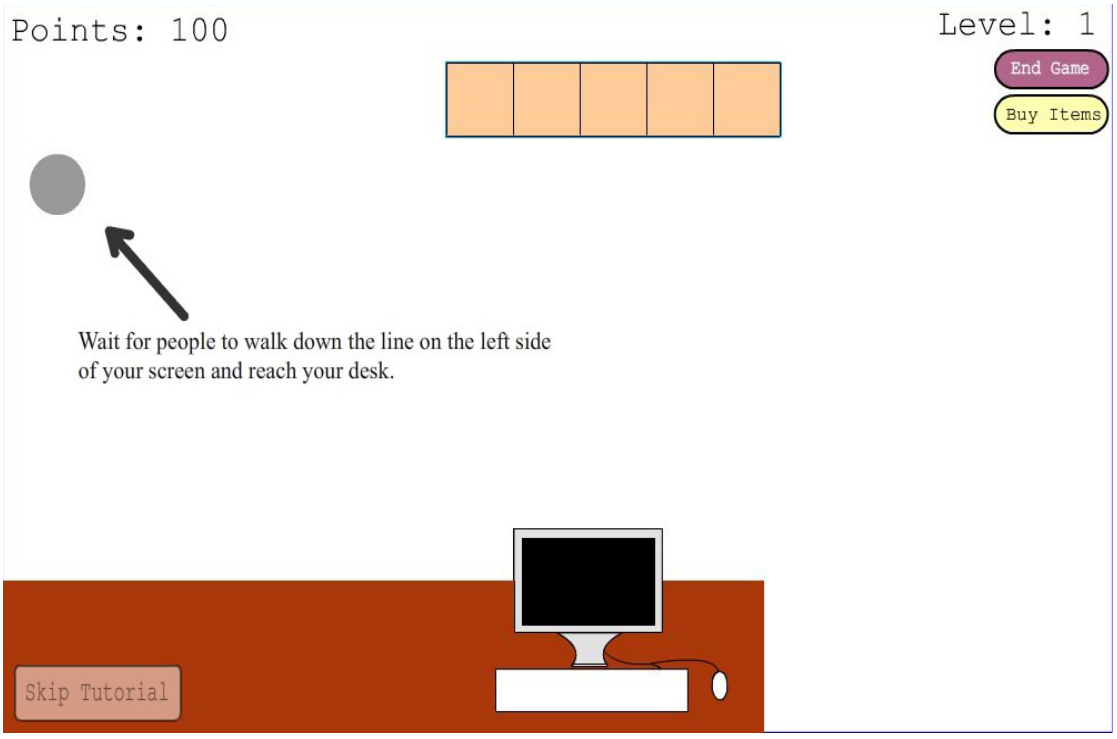


FIGURE 4.3: Tutorial Images - Employee Movement

they form a line and wait for their devices to be registered and allowed to be on the company's network. The player can register a device by clicking on its employee and sending him a form to fill out (Figure 4.4), which he will, in turn, send back to the player. Once this action has been completed, the employee will turn green, showing that his device has been authorized and earning the player 5 points. Then, he will move to a random location of the address space, which is located in the top-center of the screen (see Figures 4.5 and 4.6).

However, like in real life, some employees are too impatient or are too busy to wait for the admin to authorize their devices to be on the network, so they decide to connect their device to the network themselves. In Device Dash, this is shown by employees changing shades of color from gray to purple, to signal their frustration to the player. If they are frustrated enough, represented by the color purple, employees will leave the line, go to the address space, plug their devices into the network, and make the player lose 5 points. This is represented by their picking a random address (box) in the address space and staying there for a certain amount of time, which is also randomly defined. Figure 4.7 shows an employee starting to become frustrated (lavender color), while Figure 4.8

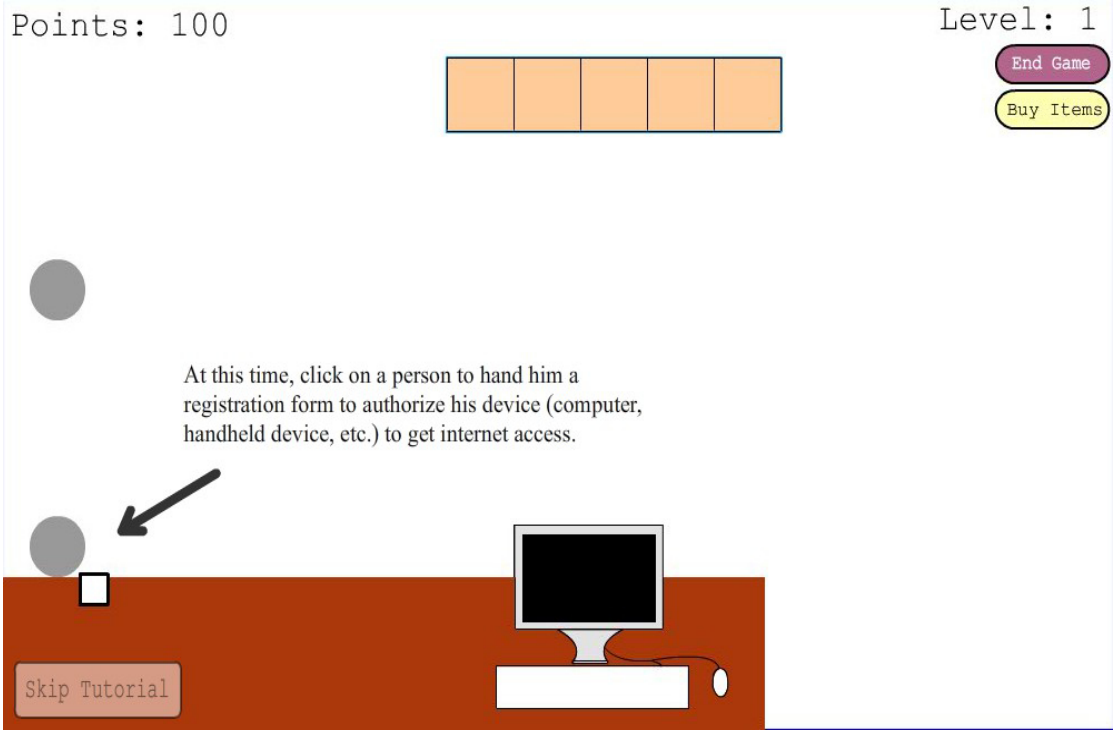


FIGURE 4.4: Tutorial Images - Registration Process

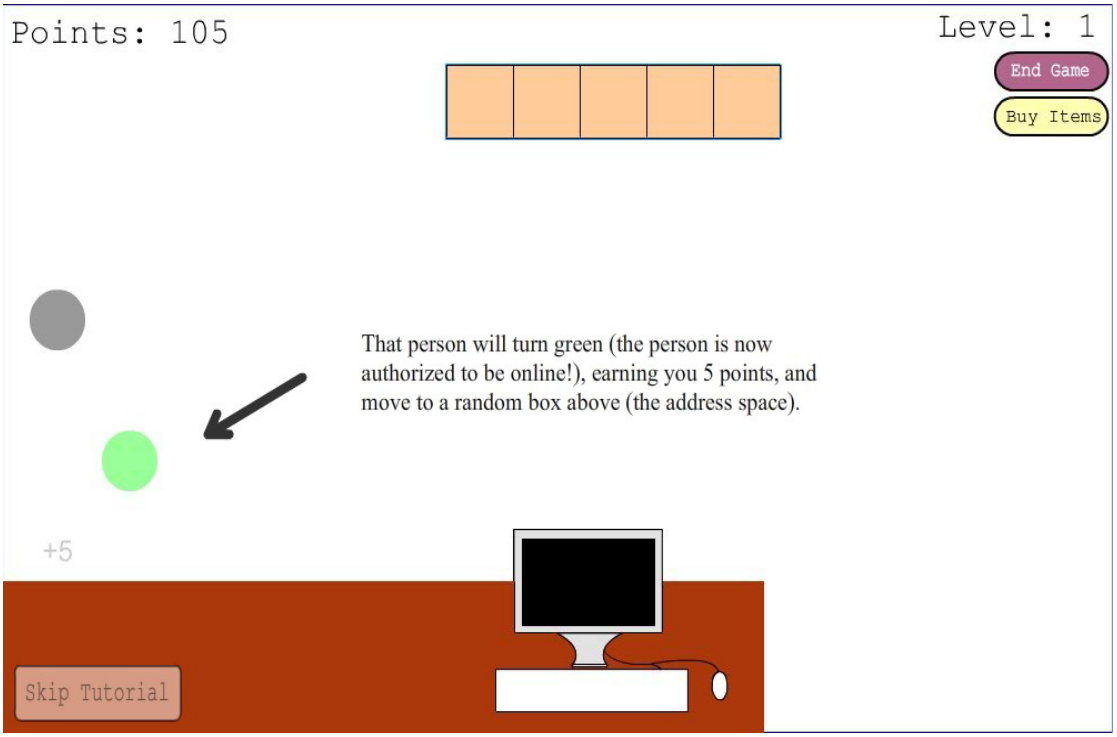


FIGURE 4.5: Tutorial Images - Authorization Color Change

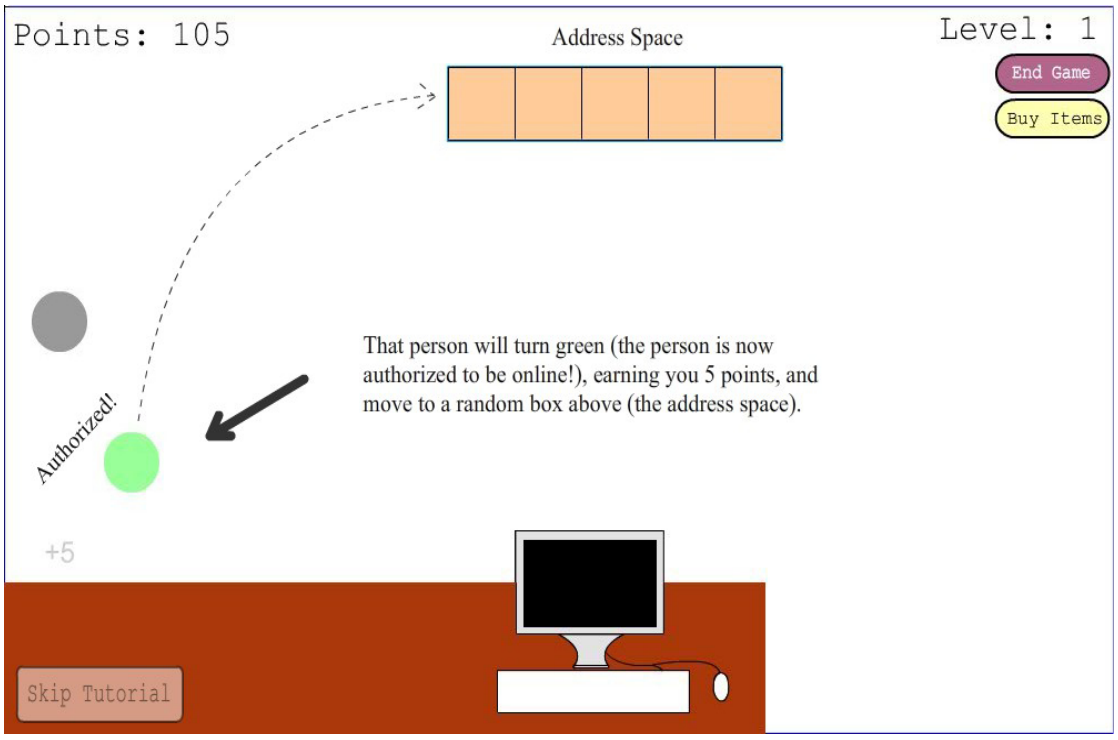


FIGURE 4.6: Tutorial Images - Authorization Movement to Address Space

shows the movement of the employee when he goes to the address space without being authorized to do so.

Once an employee has moved to the address space without being registered by the admin, his device becomes unauthorized and changes to gray in the address space. Devices that are unauthorized have the potential to become compromised and turn red, as shown in Figure 4.9. They are especially dangerous because, once they turn red, they will make the player lose a greater quantity of points with time. In real life, this is even more risky, since compromised devices have been attacked and may contain viruses or backdoors to the device's information.

If the player finds any unauthorized or compromised devices in his address space, he should remove them immediately, in order to earn the maximum amount of points, but should leave any authorized devices in the address space alone. The removal of an unauthorized, but not compromised devices, will earn the player from 10 to 0 points, depending on how long it took the player to remove the device, compared to the time when the device joined the address space. The removal of an authorized device will make the player lose 10 points. Leaving a compromised device on the network will make the player lose points, depending on the amount of time that the compromised device was assigned to stay on the network for and when, during this period, it was removed. This loss will be reflected when the player removes the device. Removing a compromised

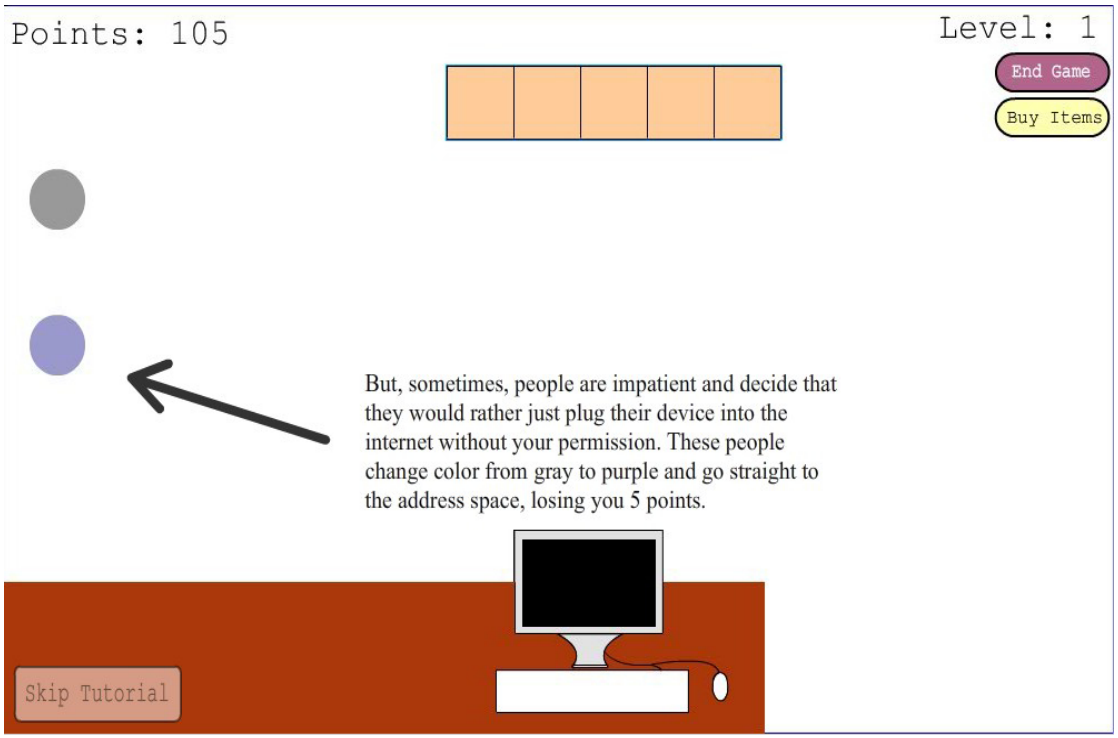


FIGURE 4.7: Tutorial Images - Impatient Employees Change Color

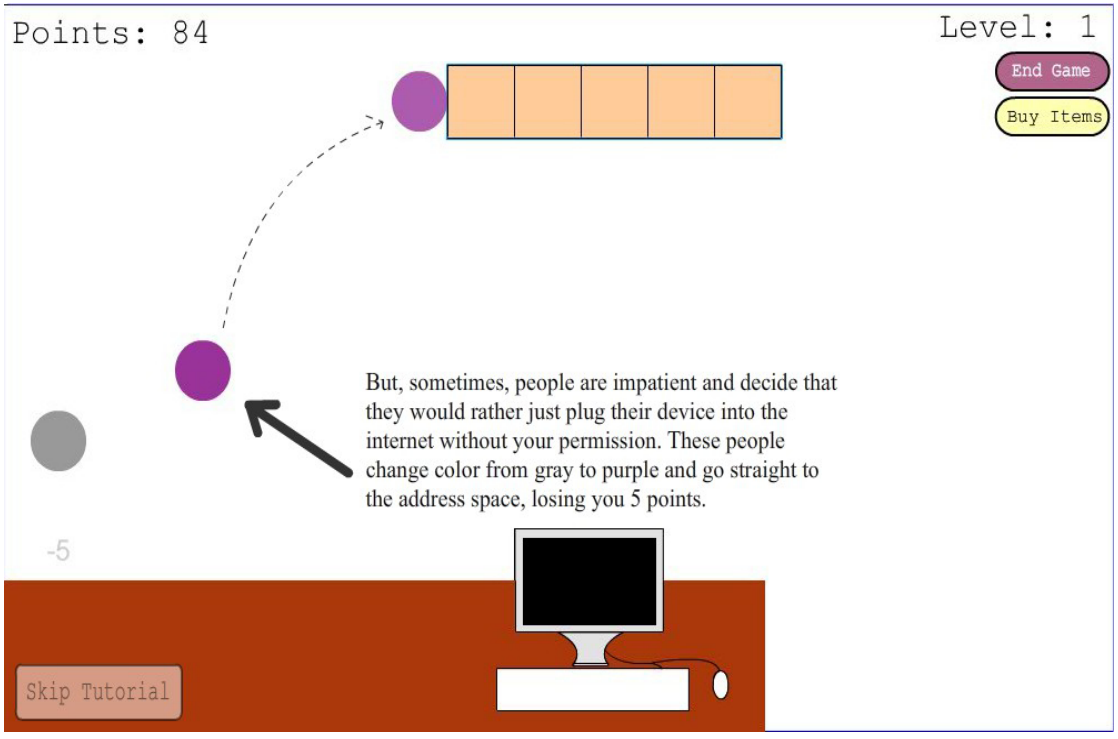


FIGURE 4.8: Tutorial Images - Unauthorized Employee/Device

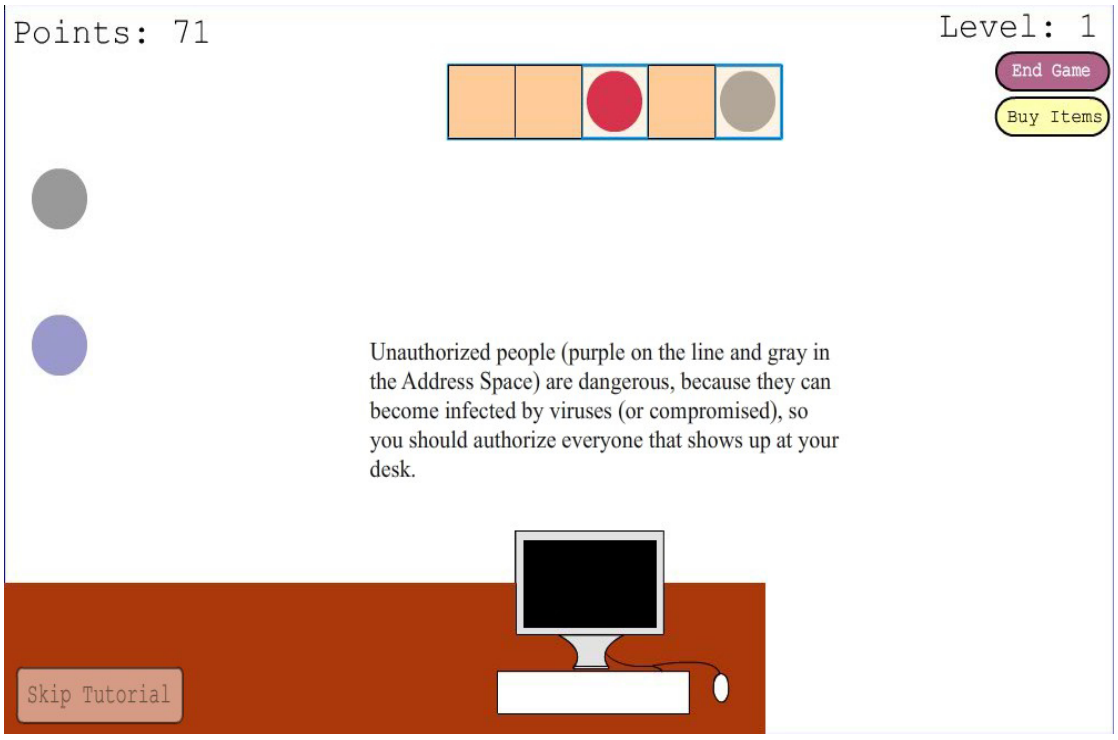


FIGURE 4.9: Tutorial Images - Unauthorized and Compromised Devices in the Address Space

device will not earn the player any points, but it will make him lose less points than if the device leaves on its own.

To search for unauthorized or compromised devices in the address space, the player must click on an address box. The box's cover will fade out, revealing the current state of the device in it, if there is a device. The cover will wait, then it will fade back until it is semi-transparent and the item in it is partially visible. To remove a device from the address space, the player must click on the device while it is still visible and the cover of the box is not. Translucence will allow the player to see the device in the box, but it may also hide information about the device. If the device in the box changes state, from unauthorized to compromised, for example, and the player does not re-click on the box, he will not be able to see this change. The idea is that, if the player decides not to remove a device in the address space, he can keep track of its last state. For example, let us assume that the player clicked on a box and the device in that box was unauthorized the last time that the player checked it. Once the box's top becomes translucent, the device within the box may have become compromised. However, the player will only be able to see the last checked state of the device, which was its unauthorized state (gray color), through the translucent box top. Hence, to check the current status of the device, the player will need to frequently patrol the address space, checking all of its boxes. Of course, the boxes that contain authorized devices will not need to be rechecked

in the lower levels, since, once they have been authorized, they cannot be compromised. This is something that is not true in real life, but that is assumed as being true in Critical Control 1. See Figures 4.10 and 4.11 for a more visual explanation.

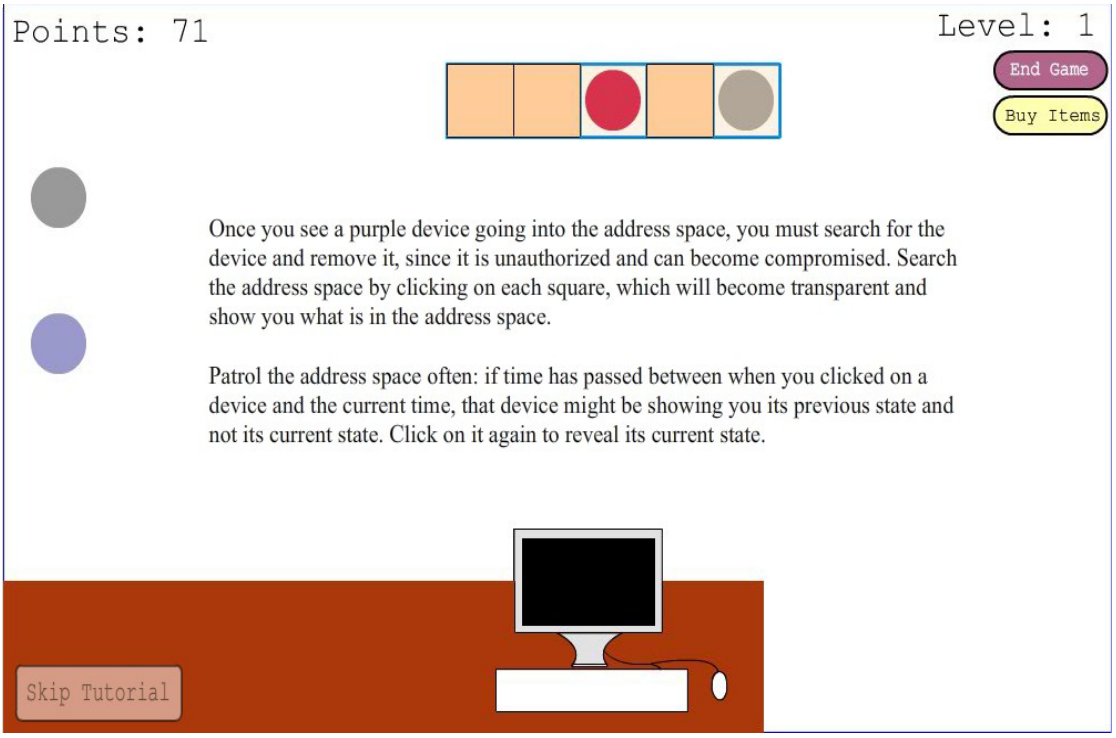


FIGURE 4.10: Tutorial Images - Unauthorized and Compromised Devices in the Address Space 2

4.3 Gameplay, Items, and Purpose

There are seven levels in Device Dash. During each level, the player needs to make choices according to the rules and purpose of the level. In order to help the player make the best choice for each situation, a number of messages are available that hint at what the player should implement. In addition, to make the game more fun, the player is allowed to buy items that can help him to better succeed at his job and gain more points. In the following, we describe the tools and messages associated with each level. A complete list of each level's information can be found in Table 4.1.

4.3.1 Description of Level 1

Figure 4.12 represents the game's interface at the beginning of level 1.

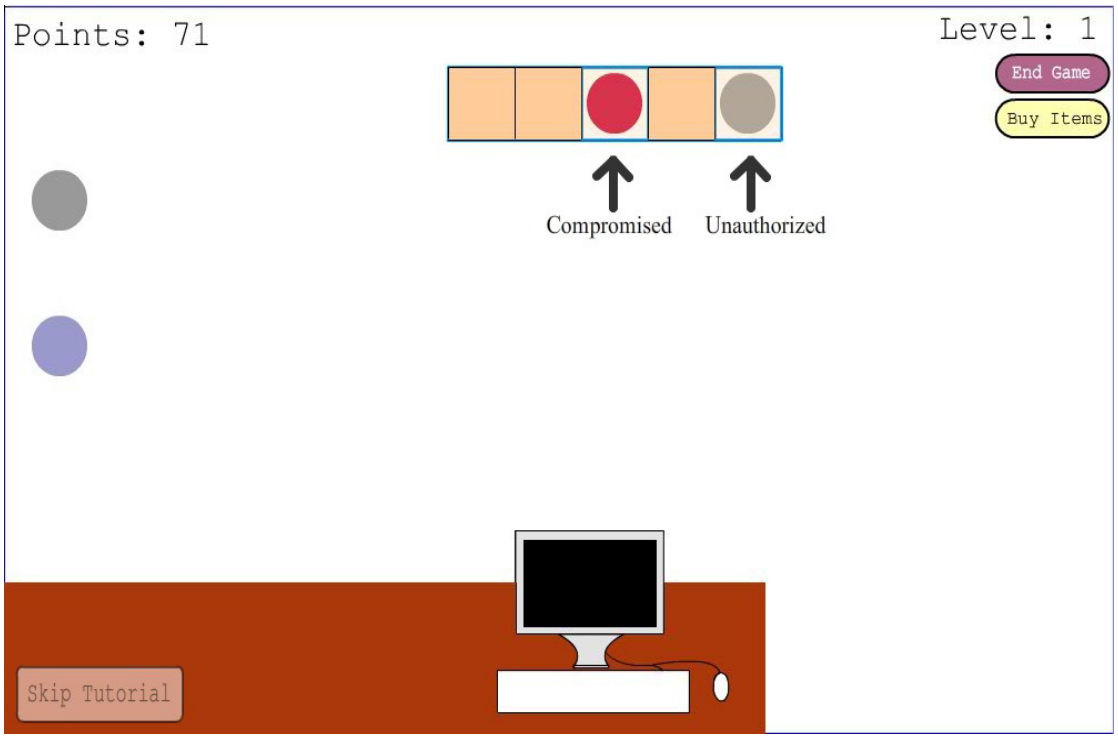


FIGURE 4.11: Tutorial Images - Unauthorized and Compromised Devices Differences

Level	Size of Address Space	Speed of Level	Items	Admin	Infection
1	5	Slow	N/A	no	no
2	15	Medium-slow	Hourglass	no	no
3	25	Medium	Scanner lvl1	no	no
4	25	Medium	Wall 1	yes	yes
5	30	Medium-fast	Scanner lvl2	yes	yes
6	30	Medium-fast	Wall 2	yes	yes
7	30	Fast	NAC	yes	yes

TABLE 4.1: Level by Level Information

4.3.1.1 Gameplay

During this slow and easy level, the player will learn how to play the game. In addition, he will understand more about Critical Control 1's capability and risk metric concepts.

4.3.1.2 Purpose

The player will learn about authorized and unauthorized devices. In addition, he will be able to see the results of not patrolling the whole address space (idea of coverage): any unauthorized device that exists in that space and is not being patrolled can become

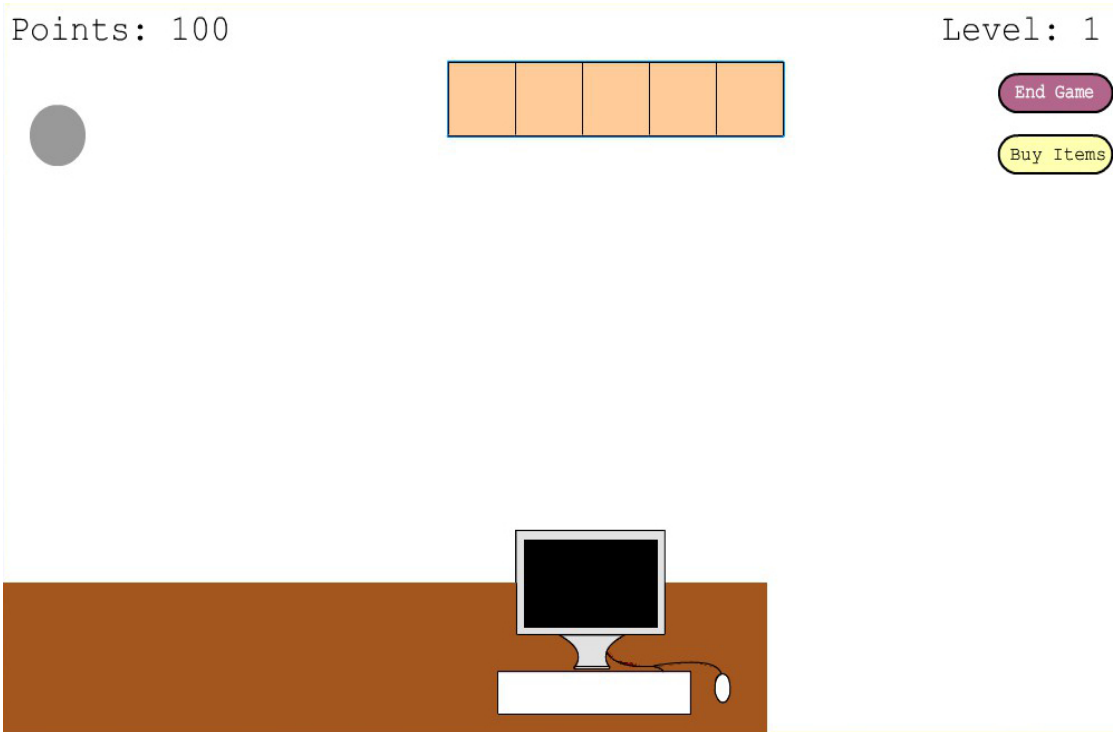


FIGURE 4.12: Level 1 Gameplay

compromised and result in the player's losing a significant number of points. Furthermore, he will be able to understand the importance of timeliness, which will result in his either gaining more points, if he patrols the whole IP address space and removes unauthorized devices in the shortest amount of time possible, or in his losing many points, if he is very slow at removing compromised devices. The player will also get a feel for the operational metric presented in CC1: each device will either become compromised or will not, depending on a randomized variable. The devices that will become compromised have a certain timeframe between when they are unauthorized and when they become compromised. If the player does not remove the device quickly enough once it has been compromised, then he will lose points, depending on the amount of time that the device has been compromised. This loss of points will make players more aware of both the timeliness of patrolling the address space and how to deal with compromised devices (i.e.: to remove them immediately).

4.3.1.3 Items

During level 1, the player will not be able to buy any items.

4.3.2 Description of Level 2

Figure 4.13 depicts the computer screen at the beginning of this level.

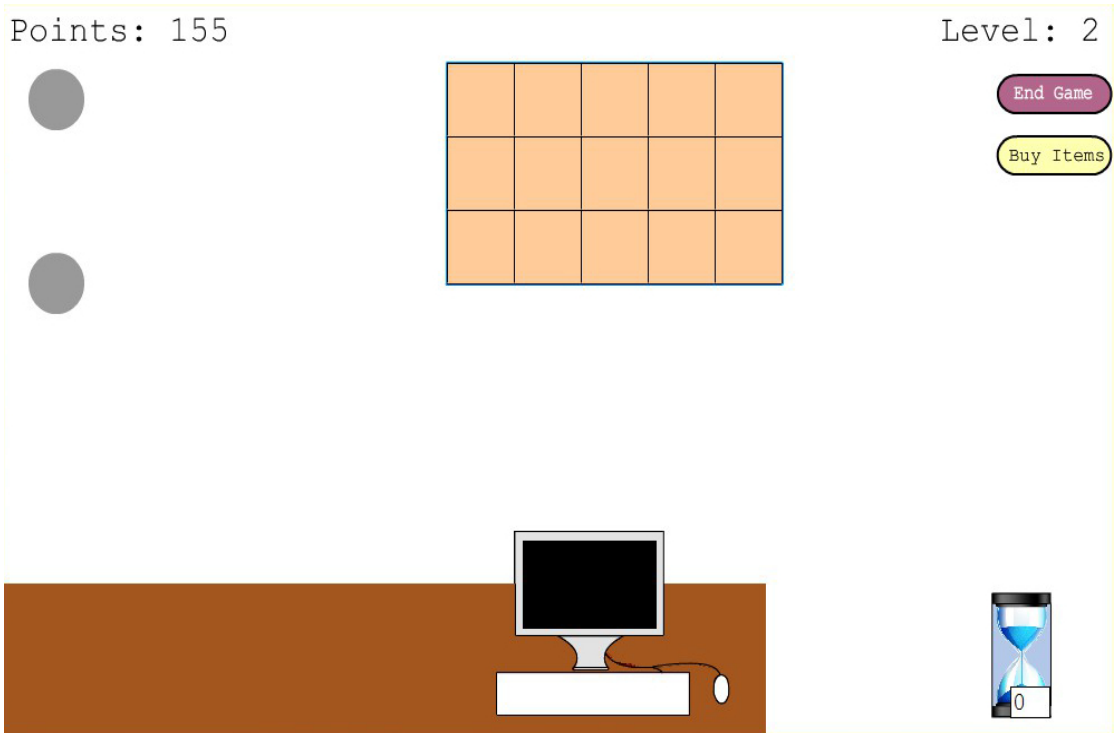


FIGURE 4.13: Level 2 Gameplay

4.3.2.1 Gameplay

At level 2, the speed with which employees move to the player's desk increases. At the same time, the address space increases from 5 spaces to 15 spaces. These changes are not very crucial, and the player should get used to them fairly quickly. With each level, the rate at which people become frustrated increases, so there are more people moving from the line to the address space, in a given time frame in the higher levels compared to level 1.

4.3.2.2 Purpose

Changes to the game format will prepare players for the later, faster, and harder levels. They will, with the increase of speed, find out what strategies to use in order to beat the game.

4.3.2.3 Items

Players will have the ability to improve their speed, by buying hourglasses. Hourglasses will reduce the speed of the employees' movement down the line to be 5 pixels per second for 5 seconds, which is a 50% decrease in speed, at level 2, but is a greater percentage decrease in speed at higher levels. Hourglasses provide temporary help with timeliness.

4.3.3 Description of Level 3

Figure 4.14 represents the computer screen at the beginning of this level.

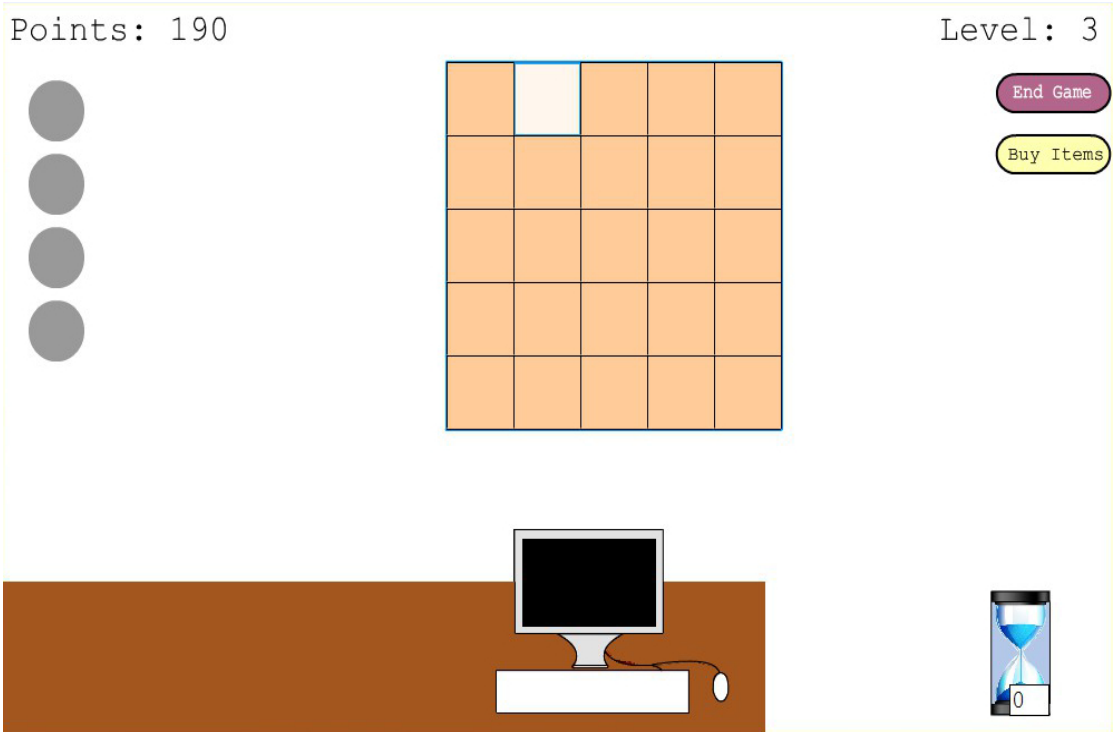


FIGURE 4.14: Level 3 Gameplay

4.3.3.1 Gameplay

At level 3, the players have to deal with an increase in the size of the address space, which now has 25 boxes. In addition, the speed of the line also increases. However, no other distractions are implemented in this level.

4.3.3.2 Items

In order to get the player used to level 3, he will receive a free Bronze Scanner. This scanner patrols the address space at a pace of one space per second, uncovering the spaces from top to bottom, left to right, one at a time, and shows the player what each space contains. The scanner only reveals, but does not remove devices. Therefore, the player still has to remove unauthorized devices.

4.3.3.3 Purpose

In this level, the message is to understand how the scanner works and why it is implemented. The scanner fills in the gap of the manual scanning of the network: by this level, the player has noticed that the address space will continue to increase in size and that having to manually scan it will not be easy in the higher levels. The need for an automated and continuous monitoring system, discussed in Critical Control 1, should be evident by now. Using the scanner will show the player how much easier his job could be.

4.3.4 Description of Level 4

Figure 4.15 represents the computer screen at the beginning of this level.

4.3.4.1 Gameplay

At level 4, the speed of the line and the size of the address space has not changed, but that does not mean that the number of challenges has stayed constant.

Computer administrators are introduced to the game. In real life, admins have a wide range of jobs, from supervising a section of the network to fixing errors on devices. There is an evident need for multiple admins in a network, which is why they are part of the game. However, admins are also a pesky part of the game because, since they have more privileges than other users, they are also prone to do more damage to the network. If a non-admin device becomes infected with malware (or compromised) it will stay on the network or it will leave the network, but will not infect other devices in the game. However, if an admin gets infected, it will spread that malware to any devices that are in the same address space. There is only one way to assuage this, which is not a definitive remedy, but it is an option: the player can use walls to partition the address space into sections. Each of these sections, then, becomes a separate address space. In

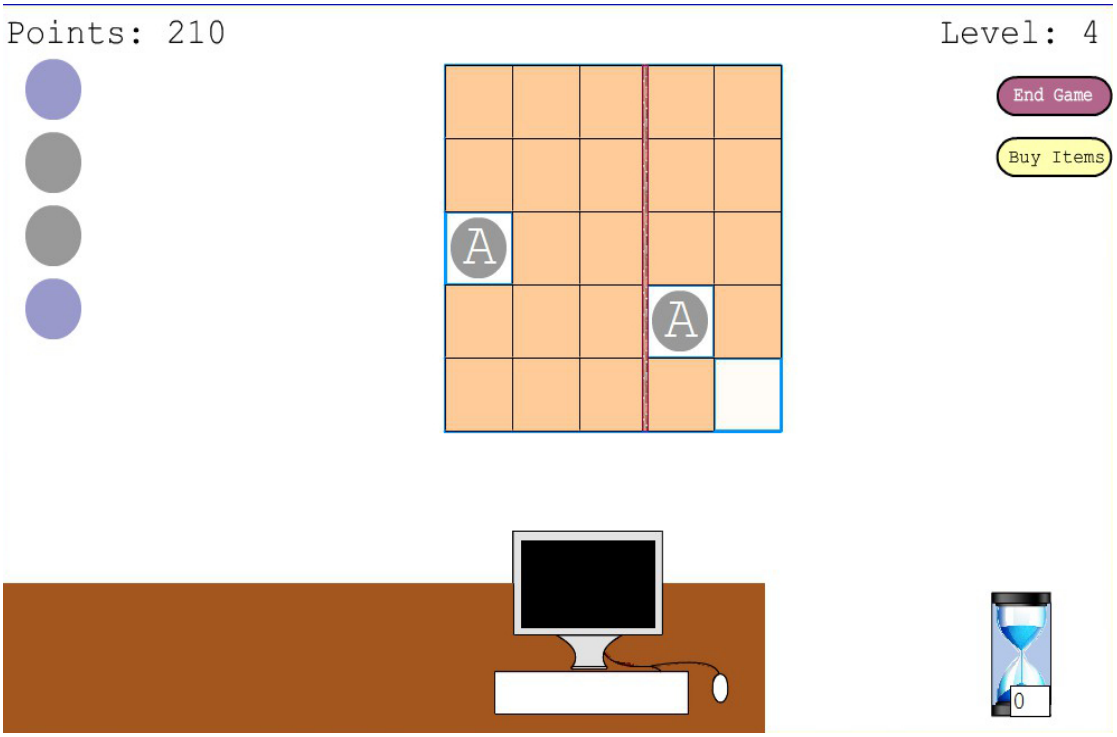


FIGURE 4.15: Level 4 Gameplay

Device Dash, no infections will be able to spread from one subnet to another. When an admin becomes compromised in a subnet, all of the admins in the other subnets will flash a shield (see Figure 4.16) to indicate that they have been protected by the wall(s).

FIGURE 4.16: Zoomed-In Image of Admin Protected from Infection by Wall

As stated in Section 4.3.1.2, if there are compromised devices on the network, the best practice is to remove them immediately. However, if there is a compromised admin in a subnet, then all devices, authorized and unauthorized, will become infected by the admin. Although the player might think that removing the cause of the problem, the admin, would be the best solution, this is not always the case. For an infection

to be completely removed from a subnet, Device Dash requires that all compromised devices be removed before the compromised admin is removed, at which point, a new non-compromised admin will appear at a random location in the subnet. If the admin is removed first, and a new admin appears at a random location of the subnet, the surrounding devices will continue to compromise the new admin, and any new admins that appear, due to the infection still residing in the subnet. There is a need for at least one admin per subnet.

Removing the admin will cost the player a deduction of 50 points. So, removing the admin and the surrounding devices will cost the player a deduction that is greater than 50 points from his total score.

4.3.4.2 Items

At this level, the player is introduced to walls, which split the address space into subnet, and admins, which have the ability to spread infections and compromise everyone in the subnet. Admins appear randomly on the address space and do not cost anything.

At the beginning of this level, the player will get a free wall that splits the address space vertically into two subnets. An upgrade can be purchased at higher levels.

4.3.4.3 Purpose

The idea in this level, as mentioned in Section 4.3.4.1, is to teach the player about the spread of infections across subnets due to admins (Critical Control 8), as well as how to use boundary defenses (walls, in the case of Device Dash) to prevent the spread of infections across a network (Critical Control 5). Figure 4.15 presents a visual representation of this level.

4.3.5 Description of Level 5

4.3.5.1 Gameplay

The address space has increased to 30 spaces, as shown in Figure 4.17, the speed of the line to the player's desk has also increased, admins are still being used, and infections from admins are possible.

The player should use this level to get used to relying more and more on his items - scanner, wall, and hourglasses - to help him through to the next level.

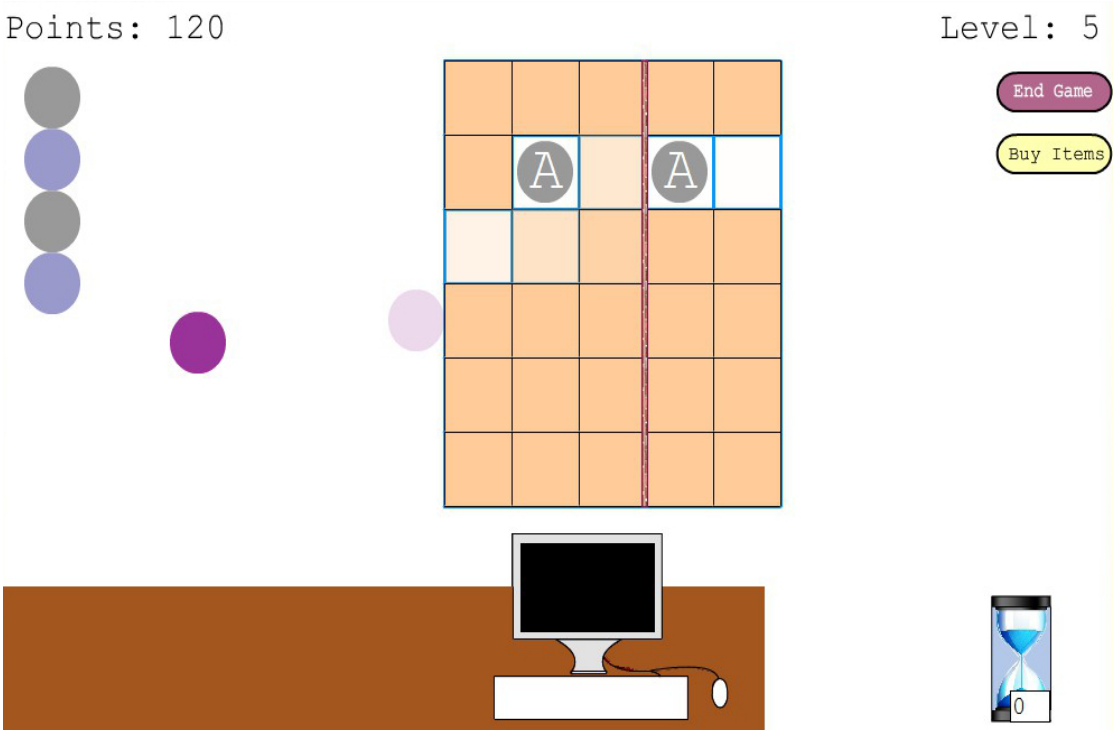


FIGURE 4.17: Level 5 Gameplay

4.3.5.2 Items

To help overcome the increased degree of difficulty, the player has the possibility to purchase an upgrade for his current Bronze Scanner, a Silver Scanner, which scans one space of the address space every 0.25 seconds.

4.3.5.3 Purpose

Players should learn about optimizing scan times by using a faster scanner, as well as get accustomed to admins and their spreading of infections.

4.3.6 Description of Level 6

4.3.6.1 Gameplay

At level 6, the speed of the line and the size of the address space stay the same as in level 5, in addition to the presence of admins and the spread of infections. Because the speed is very fast and because of the increase in the rate with which people become impatient (purple), this level is harder than level 5. See Figure 4.18 for a depiction of this level.

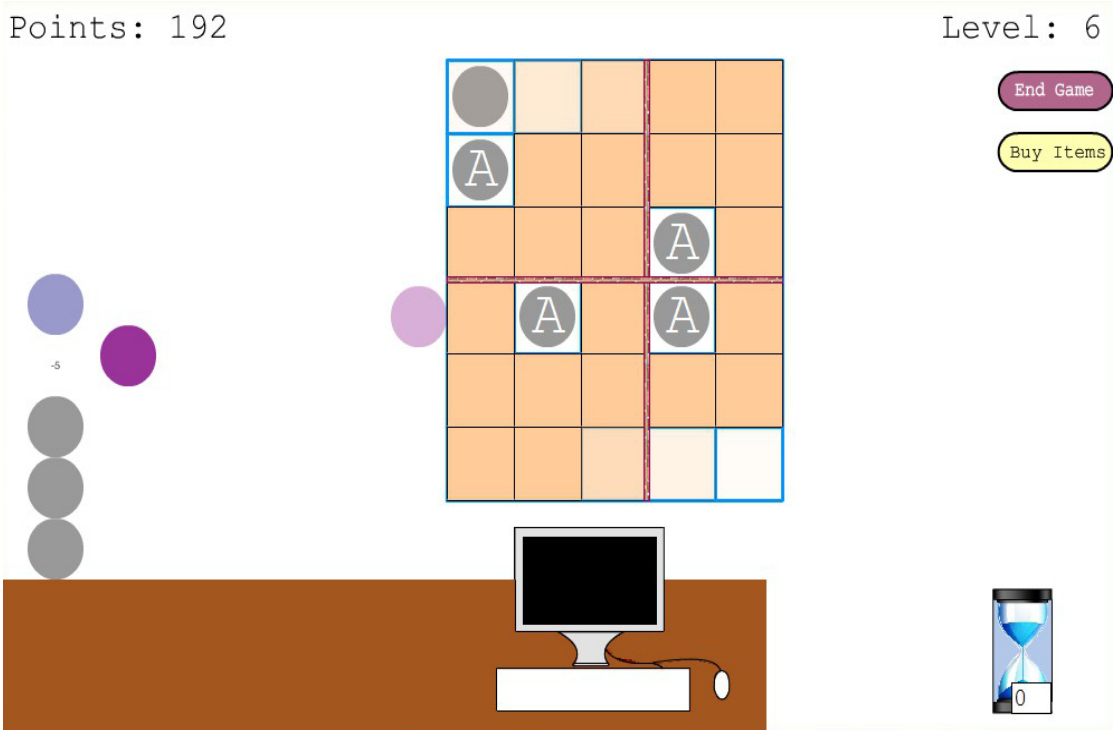


FIGURE 4.18: Level 6 Gameplay

4.3.6.2 Items

Because of the difficulty of this level, the player is allowed to buy a second wall, which will be placed horizontally in the center of the address space, creating 4 subnets. These four subnets will make it easier for the player to keep track of infections and remove them in a timely manner.

4.3.6.3 Purpose

By using the walls and splitting the address space into subnets, the player will learn to optimize his searching and removal time of unauthorized devices from the network. He will understand the full impact of Critical Control 1, the main purpose of which is to understand the importance of having an automated and constant monitoring of the network.

4.3.7 Description of Level 7

Figure 4.19 represents the computer screen at the beginning of this level.

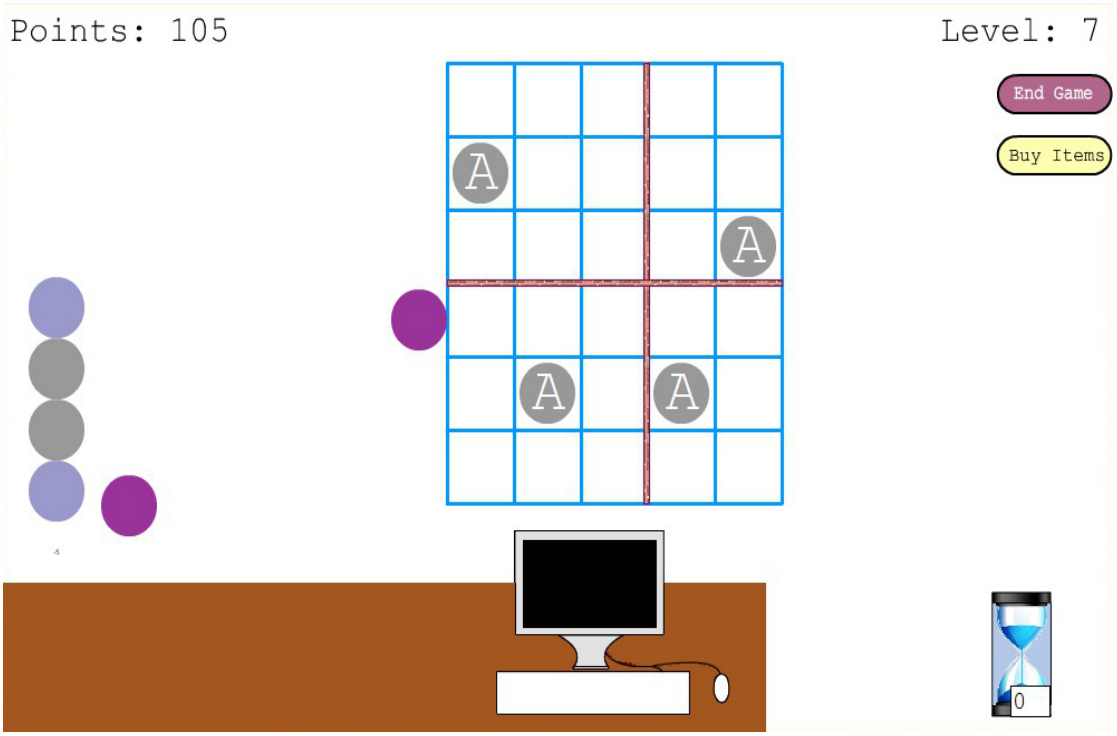


FIGURE 4.19: Level 7 Gameplay

4.3.7.1 Gameplay

At this level, the speed of the line will increase, while the address space will maintain its previous size.

4.3.7.2 Items

There is only one item that the player can buy at this level, and it is the most important item of them all. NAC (Network Access Control) is used to keep unauthorized devices (and devices that are not up to date with updates or do not have the requirements necessary to be on the network) off of the network. This is crucial in Device Dash because, by keeping out unauthorized devices, infections are kept at bay. This means that the process of removing unauthorized devices from the address space has become automated, as well as continuous, making a great difference in the job of the player, who now only has to worry about authorizing devices to be on the network.

4.3.7.3 Purpose

The message of this level is similar to that of level 6, in which the player will understand the importance of an automated and continuous monitoring system. Furthermore, by this level, the concepts of Critical Control 1, 5, and 8 will have been understood and internalized by the player.

Chapter 5

Game Implementation

"The ideology of a game is in its rules, its invisble mechanism, and not only in its narrative parts." Paolo Pedercini

5.1 Drawings, Frames, and Code

For Device Dash, we decided to use Adobe Flash Professional CS5.5 with Action Script 3, since a great number of games are done in Flash, not to mention that we wanted to put the game online once it was completed. In addition, we wanted to have the ability to easily and seamlessly draw and code in the same program, which Flash gave us. Furthermore, Flash is also great for animations, which we needed to incorporate into the game.

5.2 Code Implementation

Since Flash's Action Script 3 is object oriented, its code structure is similar to Java's, in which each object has a designated class. Each object drawn and defined in Flash can be linked to a class, where functions that pertain to the object are defined.

Device Dash's code uses the basic game structure as defined in Gary Rosenzweig's *Actionscript 3.0 Game Programming University* [8]. This structure includes a main class for the game, whose most common function is a game loop that takes care of updating the game based on the game's changing frames and multiple other classes that are defined for items that need to have functions. For example, the Person item needs to have variables that keep track of its current status (authorized, unauthorized, compromised, not compromised), its movement, its color, etc. and functions to change its color, when

it changes from unauthorized to compromised. There are also functions that check for the current status of the game: game is over, game is paused, and game is restarting. Then, there are functions that clean up the stage, removing all objects, and reset the game to be restarted.

Flash gives the coder two options when it comes to writing code: code can either be written in classes that are placed in .as files or it can be written in the Actions Panel. Usually, people that do heavy coding in Flash, such as gamers, write code in .as files, while people that use Flash mostly for animations with some coding, use the Actions Panel. For small projects, all of the code can be written in the Actions Panel and can be easily read. However, for larger projects, writing code solely in the Actions Panel becomes impossible, since there are too many things to keep track of. Most of Device Dash's code, for example, due to its sheer size, was written in .as files.

The Actions Panel is useful when you want Flash to perform a task that you will never have to deal with again. Buttons, for example, are a perfect example for the type of code that would go into the Actions Panel, since once the code for a button is written, it does not need to be altered. A nice feature of the Actions Panel is that it knows what objects are physically on the stage (have been dragged onto the stage), and there is no need to instantiate these objects in your code, which you would need to do for an object to be referred to in .as files. Instead, you simply refer to any object that is already on the stage. All of Device Dash's button code in the main interface was written in the Actions Panel. The whole code for the introduction was also written in the Actions Panel.

Since most of Device Dash's code was placed into .as files, we needed to find a way to grab information from each object in order to use it in the main file. There are two ways to deal with this: functions can be created in each class that deal with returning the needed variables, then manipulate these variables in the main file; or .as files can be created that will store the necessary information for the game, compute the needed results, and pass the result back to the main file. We opted for the latter approach, once we noticed the amount of data that needed to be known by multiple .as files in order to perform their respective actions.

For more information on how Device Dash was coded, see Appendix E.

Chapter 6

Testing Design

"The problem of simulating reality is that it can easily become too real."

David Wong

There are many educational games available to play, as well as literature on how to design and implement games. There is some literature on game testing, but it is not very detailed. The literature deals more with the testing results than with spelling out how the testing was executed: what questions were the players tested on, how did the players answer them, etc. Therefore, it is not easy to replicate testing done for one game on another game.

Making games is quite different from testing them: testing requires some knowledge of psychology, in order to know what to test and what not to test, as well as how to design evaluation criteria. Furthermore, it can be hard to identify what the players knew before playing the game and what they learned from playing the game.

In Section 6.1, we first outline the goals of evaluating Device Dash and describe an evaluation protocol relative to other games. Our testing is described in detail in Section 6.2. The recruitment process is explained in Section 6.2.3.

6.1 Experimental Testing Goals

As mentioned in Section 3.1, the purpose of Device Dash is to explain important parts of Critical Control 1, 5, and 8 in a more easily understandable way than by reading the SANS' description of the Critical Controls. We also wanted players to have a more internalized and better understanding of what we were trying to teach them. Another

appeal of using a game was that the concepts would be explained in a simpler manner, without requiring the understanding of mathematical equations or computer science terms and concepts. Therefore, we decided to test how well people understood concepts taught by the game compared to reading a paper that explains the same concepts with words. Our protocol is similar to that used by Barab et al. [2]. Their paper discusses the differences between learning through play and learning through reading. They test a game that was created in order to get students to learn more about water quality problems.

Their testing phase consists of 3 different testing groups that test 3 different conditions: the expository text condition, in which students get little contextual framing and the info to be learned is presented in a format similar to a textbook's; the simplistic framing condition, which contains one rich storyline in which content is situated, and during which the students read descriptive text about the storyline and their choices have no impact on the unfolding of the story; and the immersive world condition, in which students get to experience a virtual world, where they have to navigate an avatar in a virtual park, interview NPCs (non-player characters), and collect water quality data [2]. Their testing showed that, in the immersive world condition, players understood the concepts much better than in the other conditions and were more involved, perhaps because they were playing a game and learning without thinking about learning.

Sheng et al. [9] describe the testing of the educational security game Anti-Phishing Phil. In their paper, they do an initial test, in which they have testers play the game, then answer questions about URL composition. However, their testing showed an increase in the false positive rate, after testers played Anti-Phishing Phil. This was because testers misinterpreted the URLs that they examined. They revised Anti-Phishing Phil, adding messages that Phil's father teaches him about URLs, and also revised their testing methodology. During the second testing, participants were given 10 websites and were asked to say whether each website was legitimate or not. In addition, they had to say how confident they were about each of their answers on a scale of 1 to 5, where 1 meant not confident at all and 5 meant very confident. Then, the testers had 15 minutes to complete the game's training task. After this, they were presented with 10 more websites to evaluate. Anti-Phishing Phil's testing seemed to correspond to how we wanted to test Device Dash: a question and answer format with confidence scores, comparing previous knowledge with knowledge acquired from *Anti-Phishing Phil*.

We noted that, although we wanted to see how well players understood the concepts versus readers, we also wanted to find out how confident each group was about its choices. This would help us evaluate how well the game did versus the reading. With these goals in mind, we set out to write our testing plan.

6.2 Testing Description

6.2.1 Tentative Demographics

We had few choices when it came to who to test our game on: we could test it on Wellesley College students, which meant that we would get an overly female point of view on the game. (Gagnon [4] explains how this could be a downfall because girls and boys play games differently, as well as focus on different things within the game.) We could also test it on Wellesley College faculty, and thus add a certain number of males to our testing pool to balance it out. In addition, since the game was made at MIT Lincoln Laboratory, we could test it there as well. MIT Lincoln Laboratory would give us an overly male point of view, balancing out our testing group.

We suspected one issue to come up: because the game was being tested at two scientific institutions of learning, we had a greater number of players who understood computer science concepts than who did not, and we had intelligent testers that would understand and retain the information presented to them better than the general public. Thus, our testing would be slightly skewed in favor of people who would be more prone to easily understanding the concepts that we were trying to teach them.

6.2.2 Tester Groups and Process

For our testing, we decided to have 2 groups: the Device Dash Group (or the treatment group), whose main focus would be to play the game, and the Control Group, whose focus would be to read and understand the two-page introduction to the Critical Controls. The introduction to the Critical Controls contains an explanation of the 3 Critical Controls that we wanted to teach our testers, written in a simple form that contains no equations and defines all of the mentioned computer science terms (see Appendix A). The purpose of this document is to compare its teachings to the Device Dash's teachings, which is done through a series of questions.

The tasks of these two groups are as follows.

Device Dash Group's tasks:

1. Answer 12 questions about the Critical Controls from the tester's own knowledge (knowledge-based testing).
2. Read the two-page introduction to Critical Controls 1, 5, and 8 for 10 minutes. Answer 12 questions on the reading, which are the same as the ones from 1, but are re-phrased and re-ordered (reading-based testing).

3. Read the game's walkthrough and play the game for 30 minutes (if the tester loses the game within the 30 minutes, he can replay it until the 30 minutes are over). Answer 12 questions on Device Dash, which are the same as the ones from 1 and 3, but are re-phrased and re-ordered (game-based testing).

4. Answer the 12 questions again one week later (one-week-later testing).

Control Group's tasks:

1. Answer 12 questions about the Critical Controls from the tester's own knowledge (knowledge-based testing)

2. Read the two-page introduction to Critical Controls 1, 5, and 8 for 10 minutes. Answer 12 questions on the reading (same as the ones from 1, but re-phrased and re-ordered) (reading-based testing).

3. Play *Meerca Chase*, a *Neopets* version of Snake, for 30 minutes (to make the testing process from the two groups similar, so as not to get erroneous results). Answer 12 questions (same as the ones from 1 and 3, but re-phrased and re-ordered) (game-based testing).

4. Answer the 12 questions again one week later (one-week-later testing).

All 3 sets of questions contain evaluations of the player's level of confidence in his answer to each question. These confidence evaluations would help us grade each player's learning and show us if his confidence increased with each task, in the case that all of his answers were correct. All 3 sets of questions can be found in Appendix B.

6.2.3 Recruitment Process and Location

In order to recruit players, we set up three one-hour sessions of game-play, ranging over three days for Wellesley College testers and a two-hour session for MIT Lincoln Laboratory testers. However, not much testing was done during this time; instead, most of the testing was undertaken in individualized sessions to accommodate the testers' schedules. Testing lasted from 45 minutes to 1 hour, so people could come at whatever time they wanted, as long as they had 1 hour available to test the game. We sent out emails about the testing to:

- the Computer Science Department mailing list, which includes faculty,

- the Computer Science Club mailing list, which contains non-computer science and computer science majors,

- Wellesley College's Graduating Class of 2012 mailing list
- individual Wellesley College alumnae that lived in the Boston area
- Wellesley College students that are not part of the Computer Science Department
- Wellesley College faculty from other departments
- MIT Lincoln Laboratory employees

We also set up a Facebook Event in order to find more players.

A room with many available computers was reserved at Wellesley College for the periods during which we needed to be testing. Device Dash was loaded onto 10 computers, and *Meerca Chase* was loaded onto 10 other computers. The number of testers that signed up to test the game was split into half, based on the sign up sheet, with one half being the Device Dash Group and the other, being the Control Group.

At MIT Lincoln Laboratory, 4 laptops were borrowed and a long table was set up with 2 computers having the game and 2 computers having *Meerca Chase* on them. The number of people who signed up to test the game was split into half, with one half comprising the Device Dash Group and the other, the Control Group.

The process from Section 6.2.2 was followed for the testing.

Chapter 7

Testing Results

"Please note that we have added a consequence for failure. Any contact with the chamber floor will result in an unsatisfactory mark on your official testing record, followed by death."

GLaDOS (*Portal*)

This chapter reports on the results of Device Dash's testing. Section 7.1 discusses the actual demographics of the test subjects. Section 7.2 compares the effectiveness of playing Device Dash to playing *Meerca Chase* and measure the retention of the concepts being taught for each group after one week. A detailed version of the results of our testing, such as a question-by-question analysis, can be found in Appendix D.

7.1 Actual Demographics

We had hoped that we would get a good mix of males and females for our testing. We thought that having an overly female or an overly male population of testers might hinder our testing because males and females play and view games differently, as discussed by Terlecki et al. [11]. In addition, we were afraid that we had a greater number of players who understood computer science concepts than who did not. This could also be a problem because students knowledgeable in computer science and/or computer security would know many of the concepts presented and start out with a higher score than other students. In the end, our test subjects were primarily female, the majority of which was familiar with computer science, in general, but not in particular (meaning that they were not computer science majors or very involved with computer science).

A testing group comprised of people unfamiliar with computer security concepts might have made our testing easier, because we would not have had to break pre-conceived ideas about security concepts that the SANS deemed as false in its 20 Critical Controls. However, many of our testers knew these concepts already. One such concept is that authorized devices are as likely to become compromised as unauthorized devices. However, according to Critical Control 1, only unauthorized devices are likely to get compromised. Therefore, we did have to alter some correct ideas, in the minds of our testers, in order to uphold the SANS' concepts. All of our testing questions can be found in Appendix B.

The demographics of the testers from the Device Dash Group and the Control Group can be found in Figure 7.1. (Note: RPG is an acronym for Role Playing Game, MMORPG is an acronym for Massively Multiplayer Online Role-Playing Game, and Shmups is an acronym for Shoot 'Em Up - first person shooter games. "Plays" is short for "Plays video game X on a regular basis".)

Group 1							
Name	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**
Age	21	21	21	22	22	25	23
Gender	Female	Female	Female	Female	Female	Male	Male
Plays	Perfect World, Skyrim, Animal Crossing	Call of Gods	DS + Perfect World	Elder Scrolls: Oblivion	no	no	
Plays Style	RPG/MMORPG, Action, Adventure	RPG/MMORPG, Puzzles	RPG/MMORPG, Action, Adventure, Puzzles	RPG/MMORPG, Action, Adventure, Puzzles	mmorpg	Shmups, Racing	Shmups, Action
CS major	no	no	no	no	yes	yes	no
CS knowledge	a little	yes	a little	yes	no	yes	a little

Group 2						
Name	**#8**	**#9**	**#10**	**#11**	**#12**	**#13**
Age	22	21	19	20	20	21
Gender	Female	Female	Female	Female	Female	Female
Plays	league of legends, Starcraft 2	Harvest Moon, PWI, Pokemon	Sims, LA Noir, Smash Bros	DragonVale, Pokemon	no	Kingdoms of Avnalur: Reckoning
Plays Style	RPG/MMORPG, Action, Adventure	Shmups,RPG/MMORPG, Action, Adventure	RPG, Adventure, puzzles	RPG		RPG/MMORPG, Adventure, Puzzles
CS major	no	no	no	no	no	no
CS knowledge	no	a little	yes	a little	no	a little

FIGURE 7.1: Demographics - Group 1: Device Dash Group and Group 2: Control Group

	Actual	Percent Value
Avg. Score for Device Dash Group (/12)	7.93	66.07
Avg. Score for Control Group (/12)	6.58	54.86
Avg. Confidence for Device Dash Group (/60)	31.43	52.38
Avg. Confidence for Control Group (/60)	29.33	48.89

TABLE 7.1: Knowledge-Based Testing Data

Figure 7.1 shows that there are seven testers in the Device Dash Group that played Device Dash and 6 in the Control Group that played *Meerca Chase*. It is noticeable that 9 out of 13 testers play video games on a regular basis and that most of our testers have played or play video games that deal with a speed factor, which could, potentially, make it easier for them to play Device Dash, since it requires speed and concentration.

We had 11 female testers and 2 male testers, whose ages ranged from 19 to 25 years. Their testing results are discussed in the following section.

7.2 Testing Results

We gave each person, based on his time of arrival to the testing location, a group number, making sure that each group's members were randomly selected.

7.2.1 Knowledge-Based Testing

For the knowledge-based portion of the testing process, we asked 12 multiple choice questions to each tester, who had to answer each questions as well as tell us, on a scale of 1 to 5 (1 being not confident and 5 being very confident), how confident the tester was about his answer. The scores and the confidence measurements were added up, and for each group, an average score and an average confidence rating were calculated. Table 7.1 shows the results from the knowledge-based testing.

The knowledge-based testing provided a baseline of the knowledge of the test subjects. It revealed that the Device Dash Group had an average score of 66% correct, while the Control Group had an average score of 55% correct. The Device Dash Group, thus, scored 11 percentage points higher than the Control Group, and the confidence of both groups was around 30%. Thus, we can see that even though each group's members were randomly selected, the Device Dash Group knew more about computer security than the Control Group did. The scores would have been statistically more significant if the testing groups had been larger.

	Actual	% Value	% Change From Knowledge-Based Testing
Avg. Score Device Dash Group	10.14	84.52	18.45
Avg. Score Control Group	8.92	74.31	19.44
Avg. Confidence DD Group	48.86	81.43	29.05
Avg. Confidence C Group	44.83	74.72	25.83

TABLE 7.2: Reading-Based Testing Data

7.2.2 Reading-Based Testing

During the reading-based portion of the testing, we asked the testers to read a two-page introduction to the concepts depicted in Critical Control 1, 5, and 8 that we wanted the game to teach. The introduction comprised of relevant excerpts from the original Critical Controls simplified to avoid jargon use and to adhere more to what Device Dash teaches. A copy of the introduction can be found in Appendix A. We wanted to see how well the testers would understand the introduction, so we asked them to read it for 10 minutes, during which they could re-read it as often as they wanted to. After the 10 minutes were up, the introduction was taken from them, and a second set of 12 questions was given to the testers. This set of questions was, content-wise, the same as the previous set, but the questions were reordered and their wording was changed, to see if the testers really understood the concepts and did not just remember the previous questions and their respective answers. All of this is in accord with our remark from Section 7.2.1. Testers from the Device Dash Group maintain their higher scores and their confidence further increases after reading the introduction. Testers from the Control Group also get a score increase from reading the introduction, but their confidence level needs more time to build up.

Our calculations, as demonstrated in Table 7.2, show that the Device Dash Group did about 10% better, on average, than the Control Group did at answering the questions correctly. Furthermore, the Device Dash Group's confidence was about 7% higher than the Control Group's. We can see that, in terms of the percent of change from the previous set of questions, the Control Group improved its score by 1% more than the Device Dash Group did, while the Device Dash Group improved its confidence by about 3% over the Control Group.

When looking at individual questions, we can see that, for the Device Dash Group, there were 5 instances (or questions), over (12 total questions * 7 testers), in which the players did worse in their answers and 4 instances in which their confidence decreased from what it was in the knowledge-based part of the test. In comparison, the Control Group had 8 instances, over a total of (12 questions * 6 testers), in which they did

	Instances of Decreased Score	Instances of Decreased Confidence
Device Dash Group	5	4
Control Group	8	12

TABLE 7.3: Reading-Based Testing Data - Instances of Decreased Score and Confidence Compared to Answers of Knowledge-Based Questions

worse during this set of questions, than during the first set, and 12 instances in which their confidence decreased. For more detail, see Table 7.3. This can be explained by the fact that, even though we asked the testers to not guess the answers to questions, some guessing was involved in the knowledge-based testing. Since the testers have not yet mastery the subject, even after reading the introduction, some of the testers still display doubts and guess on their answer choices. Nevertheless, the higher the level of knowledge, the higher the level of confidence and the lower the number of instances of decreased scores.

Taking all of this into consideration, we can still say that, by reading the introduction to the Critical Controls, testers did much better, on average, as well as overall, in both answering questions and in their confidence levels. Overall, we only had one case of a negative score change per group and a single case of a negative confidence change, as shown in Figure D.2.

7.2.3 Game-Based Testing

For the game-based testing, testers in the Device Dash Group were asked to play Device Dash, while testers in the Control Group were asked to play *Meerca Chase* also dealing with speed and concentration, for 30 minutes. They could play the game they were assigned as many times as they wanted to within the 30 minutes. The only guideline that they had to follow was that they had to read the instructions before they played the designated game. After the 30 minutes of gameplay, they had answer the third set of 12 questions, which was, again, reworded and reordered. The testers could not refer back to the game to help them answer the questions.

From this portion of the testing, we found out that, on average, the Device Dash Group did about 8% (or one question) better when it came to answering the game-based questions compared to the reading-based questions. The Control Group did, on average, about 4% (half of a question) better, as shown by Table 7.4. In terms of confidence change, the Device Dash Group did about 7% better, while the Control Group did about 3% better. There were also no decreases in score or confidence from either group. All of this can be explained by the fact that, after the reading-based testing there was

	Actual	% Value	% Change From KBT	% Change From RBT
Avgerage Score of Device Dash Group	11.00	91.67	25.60	7.14
Avgerage Score of Control Group	9.50	79.17	24.31	4.86
Avgerage Confidence of Device Dash Group	53.29	88.81	36.43	7.38
Avgerage Confidence of Control Group	46.83	78.06	29.17	3.33

TABLE 7.4: Game-Based Testing Data (KBT - Knowledge-Based Testing; RBT - Reading-Based Testing)

	Instances of Decreased Score	Instances of Decreased Confidence
Device Dash Group	1	4
Control Group	5	11

TABLE 7.5: Game-Based Testing Data - Instances of Decreased Score and Confidence Compared to Answers of Reading-Based Questions

very little room for improvement, especially for the Device Dash Group. On the other hand, going from a confidence level of 4 to one of 5, the maximum level, is always psychologically difficult, as people prefer to keep some reservations about their confidence. Finally, after the game-based set of questions, the Device Dash Group is one question away from a perfect overall average score of 12. For the second group there is no reason for the confidence or for the score to have increased between the reading-based and the game-based testing, since *Meerca Chase* did not teach them anything. The only explanation is that familiarity with the set of questions has increased, which probably made the testers feel more confident. The Control Group's average score improvement consisted of half of a question, which is within the data error margin (guessing influence).

On a question-by-questions basis, the Device Dash Group seemed to have one instance of a decrease in score from the reading-based test, while the Control Group had 5 instances of decreases. At the same time, the Device Dash Group had 4 instances of confidence decreases, compared to the reading-based test, while the Control Group had 11 instances. Refer to Table 7.5 or Appendix C for more details. Device Dash's testers did better than the Control Group's testers in both their average scores and their average confidence levels.

Once the game-based testing was finished, a discussion took place between the testers, and I couldn't prohibit the Control Group's testers from obtaining information about Device Dash. As shown in the next section, this had repercussions on the results of the testing done one week later.

	Actual	% Value	% Change From KBT	% Change From RBT	% Change From GBT
Avg. Score of Device Dash Group	11.08	92.36	26.29	7.84	0.69
Avg. Score of Control Group	9.83	81.94	27.08	7.64	2.78
Avg. Confidence of Device Dash Group	53.00	88.33	35.95	34.52	-0.28
Avg. Confidence of Control Group	47.33	78.89	30.00	20.83	0.83

TABLE 7.6: One-Week-Later Testing Data (KBT - Knowledge-Based Testing; RBT - Reading-Based Testing)

7.2.4 Testing One Week Later

One week later, the testers were asked to re-answer the 12 questions from the game-based testing. 12 out of the 13 testers came back and re-answered the questions.

We noticed a 0.69% increase, on average, in the Device Dash Group's score, compared to the answers from the game-based testing and a 2.78% increase, on average, in the Control Group's score. The Device Dash Group's confidence had fallen by 0.48%, while the Control Group's confidence had risen by 0.83%, as compared to the game-based testing results. We hypothesize that this was due to the fact that the Device Dash Group's results from the game-based testing portion were very high (11/12), on average, and therefore, it was harder to obtain a great increase in the amount of points, as compared to the Control Group's scoring, which was 9.5/12 in the game-based portion of the testing. On the other hand, the discussion after the game-based testing might have influenced the Control Group's testers and increased their average score. The same can be said for the confidence rating, which decreased for the Device Dash Group from 53.29/60, in the game-based testing, to 53/60 in the testing done one week later, while the Control Group's confidence rose from 46.83/60 to 47.33/60.

From the question-by-question data, shown on Table 7.7, we can notice that there were a total of 3 instances of fallen scores for questions answered by the Device Dash Group during the testing done one week later, as compared to the game-based testing, while there were 5 instances of fallen scores for the Control Group. In addition, the Device Dash Group had 10 instances of decreases in confidence, while the Control Group had 15, in comparison with the game-based testing.

	Instances of Decreased Score	Instances of Decreased Confidence
Device Dash Group	3	10
Control Group	5	15

TABLE 7.7: One-Week-Later Testing Data - Instances of Decreased Score and Confidence Compared to Answers of Game-Based Questions

	Device Dash Group	Control Group
Knowledge-Based Testing	7.93	6.58
Reading-Based Testing	10.14	8.92
Game-Based Testing	11.00	9.50
One-Week-Later Testing	11.08	9.83

TABLE 7.8: Average Scores For Each Testing Phase

	Device Dash Group	Control Group
Knowledge-Based Testing	66.07	54.86
Reading-Based Testing	84.52	74.31
Game-Based Testing	91.67	79.17
One-Week-Later Testing	92.36	81.94

TABLE 7.9: Average Scores For Each Testing Phase (Percent)

Based on these results, we can see that the Device Dash Group did better, generally, in both the scoring amount as well as their confidence ratings.

7.2.5 Interpreting Overall Results

From Sections 7.2.1 - 7.2.4, we can see that, during each testing portion, there was an increase in the average number of correct answers, compared to the previous testing portions. However, this does not necessarily prove that Device Dash was effective in teaching better than the literature did. Hence, we created a full overview of the testing results, shown in Table 7.8, as well as a graph of its data, shown in Figure 7.2. From this data, we notice that, even though the Device Dash Group started off with a higher knowledge-based testing score than the Control Group did, the Device Dash Group still did better than the Control Group. Therefore, Device Dash did influence the Device Dash Group's increase in its average score: the Device Dash Group did better than the control group in all of the testing phases, on average, when it came to answering questions correctly.

Table 7.9 and Figure 7.3 reflect this data in terms of percentage values.

Similarly, we can see the average confidence values for both groups, during each testing phase, in Table 7.10 and in Figure 7.4. We can tell that the Device Dash Group was

FIGURE 7.2: Average Scores For Each Testing Phase

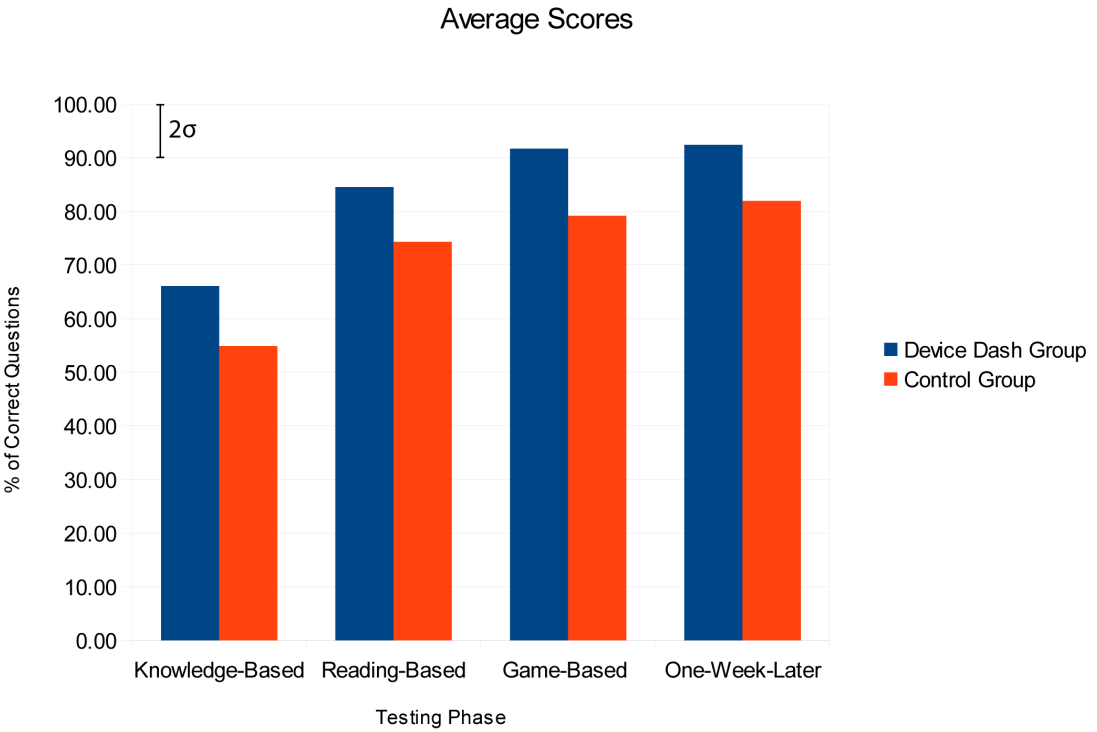


FIGURE 7.3: Average Scores For Each Testing Phase (Percent)

	Device Dash Group	Control Group
Knowledge-Based Testing	31.43	29.33
Reading-Based Testing	48.86	44.83
Game-Based Testing	53.29	46.83
One-Week-Later Testing	53.00	47.33

TABLE 7.10: Average Confidence For Each Testing Phase

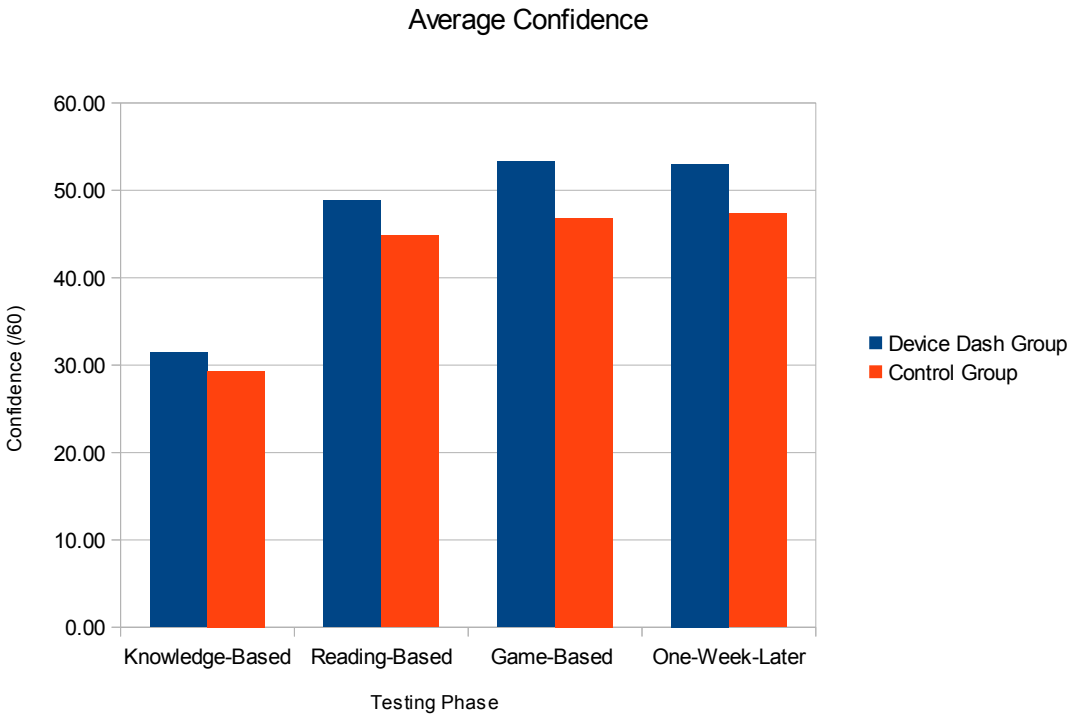


FIGURE 7.4: Average Confidence For Each Testing Phase

	Device Dash Group	Control Group
Knowledge-Based Testing	52.38	48.89
Reading-Based Testing	81.43	74.72
Game-Based Testing	88.81	78.06
One-Week-Later Testing	88.33	78.89

TABLE 7.11: Average Confidence For Each Testing Phase (Percent)

more confident than the Control Group throughout each testing phase (see Table 7.11 and Figure 7.5 for the relative percentage values).

In order to know how well the testers did during each testing phase, compared to their initial knowledge, we had to figure out how each testing phase's results compared to the players' scores from the knowledge-based testing. Therefore, we calculated the difference between the average answers from the knowledge-based testing phase and each of the three other testing phases. These results are shown in Table 7.12 and in Figure 7.6. A

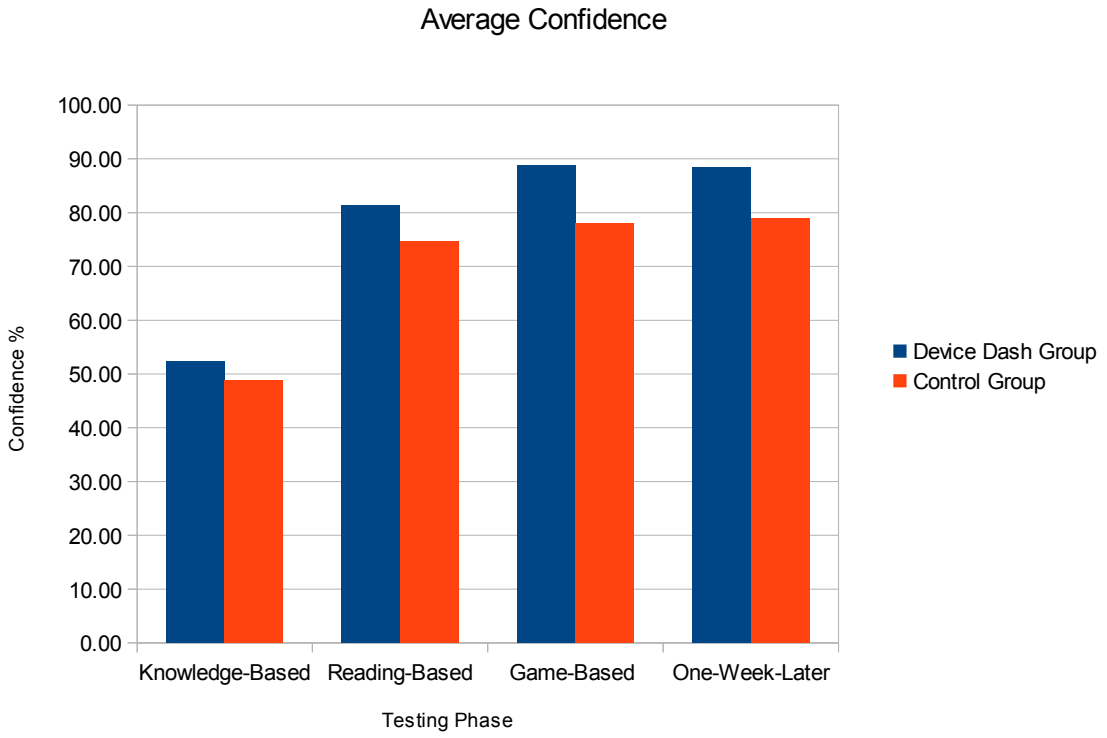


FIGURE 7.5: Average Confidence For Each Testing Phase (Percent)

	Device Dash Group	Control Group
Reading-Based Testing	2.21	2.33
Game-Based Testing	3.07	2.92
One-Week-Later Testing	3.15	3.25

TABLE 7.12: Average Score Change Compared to Knowledge-Based Testing Results

	Device Dash Group	Control Group
Reading-Based Testing	18.45	19.44
Game-Based Testing	25.60	24.31
One-Week-Later Testing	26.29	27.08

TABLE 7.13: Average Score Change Compared to Knowledge-Based Testing Results (Percent)

representation of this data using percentage change can be found in Figure 7.7 and Table 7.13. From this figure, it seems that, although playing Device Dash did increase the Device Dash Group's score, playing *Meerca Chase* also increased the Control Group's score. This surprised us because it would seem reasonable that, by playing *Meerca Chase*, a game that had nothing to do with the concepts being taught, the Control Group would not learn anything, let alone increase its score because of it. Looking more closely at the results, we hypothesize that this might have been due to an error margin, since the Control Groups score increased by about 0.5 or half of a question

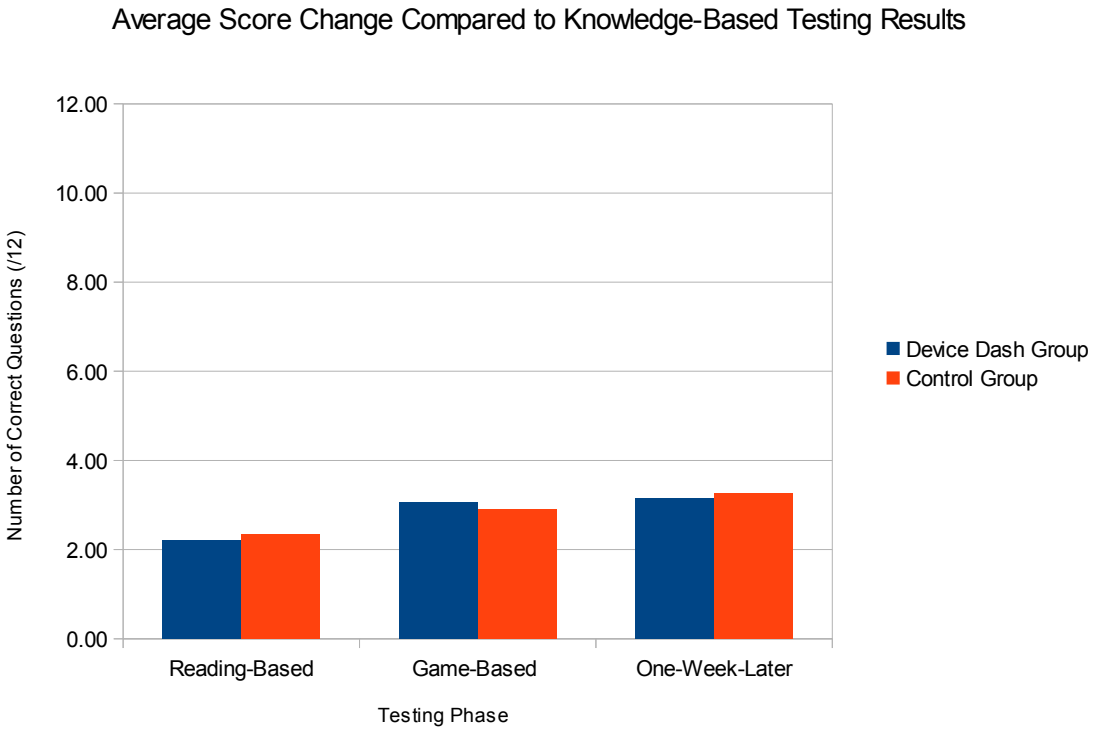


FIGURE 7.6: Average Score Change Compared to Knowledge-Based Testing Results

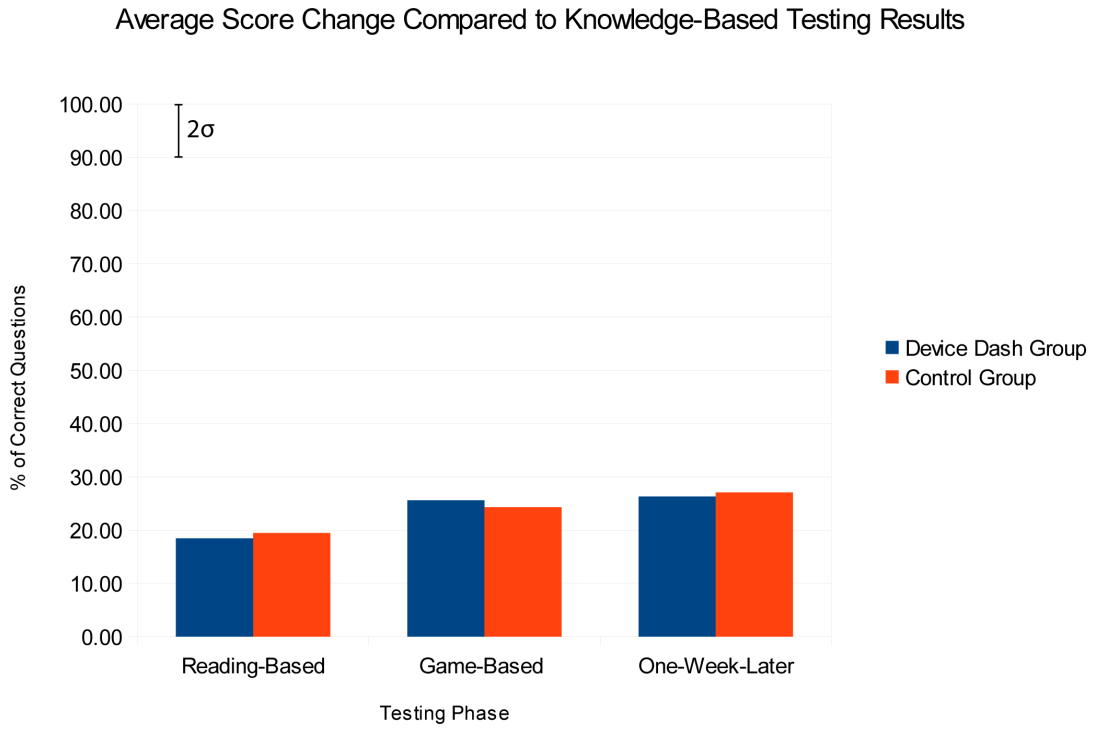


FIGURE 7.7: Average Score Change Compared to Knowledge-Based Testing Results (Percent)

	Device Dash Group	Control Group
Reading-Based Testing	48.86	15.50
Game-Based Testing	53.29	17.50
One-Week-Later Testing	53.00	18.00

TABLE 7.14: Average Confidence Change Compared to Knowledge-Based Testing Results

	Device Dash Group	Control Group
Reading-Based Testing	29.05	25.83
Game-Based Testing	36.43	29.17
One-Week-Later Testing	35.95	30.00

TABLE 7.15: Average Confidence Change Compared to Knowledge-Based Testing Results

over a total of 12 questions. Furthermore, there was a question that many people had trouble interpreting, which had two answer choices that needed to both be selected for the player to receive a full point on the question. Selecting one correct choice and one or more wrong choices would earn the player half of a point. Thus, this question might have been the culprit for most players.

In addition, another surprising outcome was that the Control Group's average score increased more than the Device Dash Group's in the one-week-later testing phase. This would suggest that the Control Group's testers learned about the game's concepts after the game-based testing ended. A possible explanation for why this happened lies in the fact that I expressed a great deal of excitement when asking people to test my game. Due to this, people were very interested in trying out the game and seeing what we had been working on during this year. Hence, when I told half of the testers that they would not get to play my game because we needed people to be in our the Control Group, most of them were very disappointed. They had put aside an hour of their time to help me out, in the promise that they would get to test my game. Once the Control Group testers finished the game-based portion of the testing, many of them started asking the Device Dash Group testers to see the game or started talking to them about the game, in order to satisfy their curiosity. When I saw that this was happening, I could not tell them to not talk about the game. This probably led to our strange one-week-later testing results showing a greater increase in the knowledge of the Control Group than that of the Device Dash Group. However, this did not affect any other testing portion.

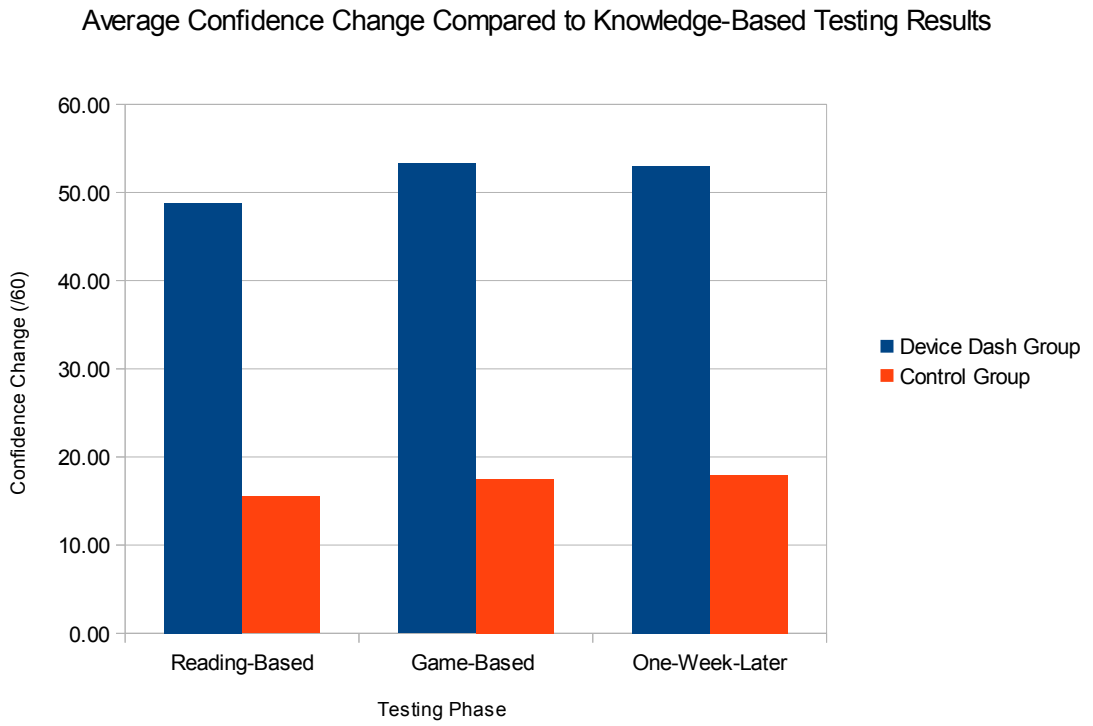


FIGURE 7.8: Average Confidence Change Compared to Knowledge-Based Testing Results

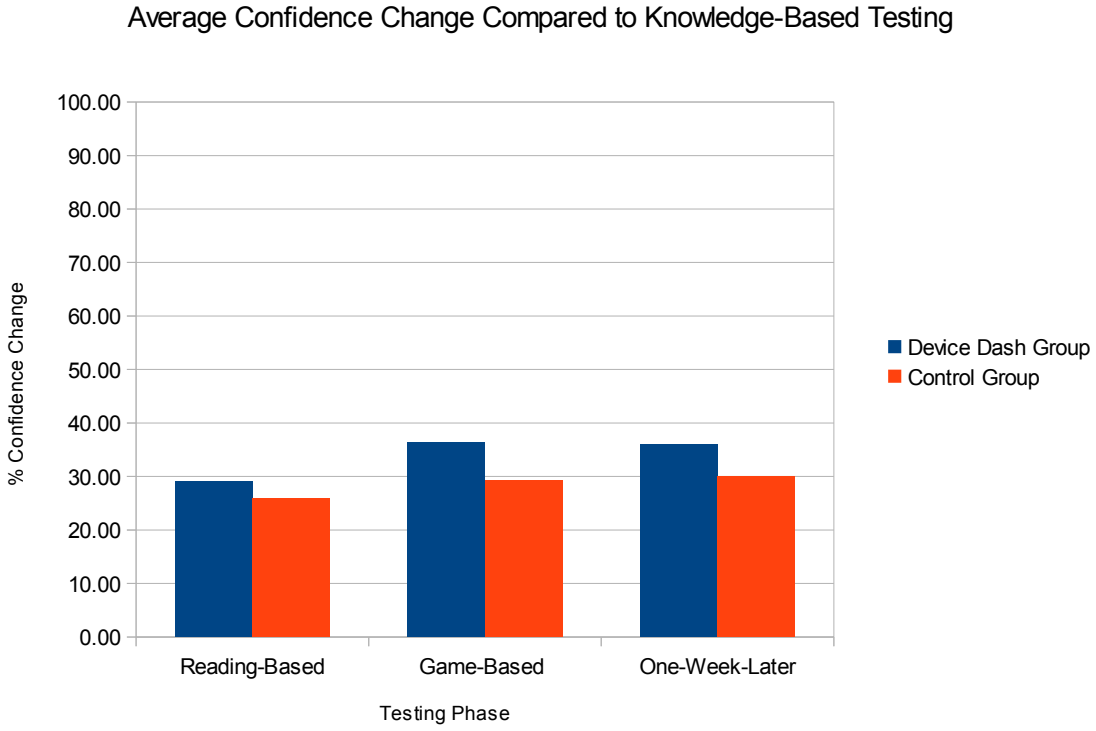


FIGURE 7.9: Average Confidence Change Compared to Knowledge-Based Testing Results (Percent)

Chapter 8

Conclusions And Future Work

"Congratulations. The test is now over. All Aperture technologies remain safely operational up to 4000 degrees Kelvin. Rest assured that there is absolutely no chance of a dangerous equipment malfunction prior to your victory candescence. [The player's platform is lowered into a fire pit] Thank you for participating in this Aperture Science computer-aided enrichment activity. Goodbye."

GLaDOS (*Portal*)

8.1 Principal Results

Computer security is an important part of today's world, where technology and data storage has become incorporated into everything from credit card information to patients' medical data. It is important to keep this data secure, since once the information has been stolen or exposed, it can never be contained again.

There is an evident need to increase awareness in computer security. Organizations such as the SANS Institute have put forth educational literature on the prevention of attacks. However, most of this literature is complicated and requires an audience that is knowledgeable in computer security. A fun and more comprehensive way to teach computer security is sorely needed. Consequently, we have developed an educational computer security game, Device Dash, which aims to do just that.

This thesis showed that our game is able to teach players about three important controls from the SANS' 20 Critical Controls in an enjoyable and retentive way. In Device Dash, the player takes on the role of a system administrator. His job is to service employees who want to connect their devices to the company's network. The admin has to register employees' devices and allow them to connect to the network. However, there are

employees that are too busy to wait for the admin's authorization and who, without authorization, join the network. The admin's tasks expand to patrolling the network's address space and making sure that all unauthorized devices are not allowed to be on the network. Device Dash's difficulty increases with each level as speed increases along with the addition of new challenges. We have provided a detailed level-by-level overview of Device Dash in Chapter 4, along with a description of the game's teaching goals, its gameplay, and the items available in each level.

Device Dash was implemented using Adobe Flash Professional CS5.5 in conjunction with ActionScript 3. Flash was selected because it gives its users the ability to both code and animate on the same interface. The code was written from scratch without any reliance on game engines, and most of the graphics were drawn from scratch with very few stock photos and images being used.

In order to assess Device Dash's effectiveness in teaching computer security, we designed an evaluation process involving two groups of players, the Device Dash Group and the Control Group, and four different testing components: (1) a knowledge-based test, (2) a reading-based test, (3) a game-based test, and (4) a one week later test. Although we observed a net improvement of the Device Dash Group's average scores after playing the game, we expected the Device Dash players to have a much bigger improvement compared to the Control Group. There are many explanations for this.

One reason could be that each group's sample size was too small; a bigger sample size could have resulted in a more decisive outcome for our game. Due to the small sample size and random chance, the Device Dash Group performed better on the tests from the start, even before playing the game. This made the Device Dash Group run into "ceiling" effects sooner, where the players could not improve their score as much since they started from a high baseline. During the reading-based testing phase, the testers were given a short and easy to understand introduction to the parts of the three Critical Controls concerned by the game, which already explained everything that we wanted to explain in Device Dash. When reading the introduction, our smart Wellesley College and MIT Lincoln Laboratory testers absorbed its information, understood it fully, and retained it very well, which led to their doing very well on the reading-based testing. The high average score for the Device Dash group after the reading-based testing phase made it extremely difficult for the testers to drastically increase their learning in the next testing phase, the game-based testing phase. After one week, our testing showed that the average score for the Device Dash Group was about the same as the game-based testing's. In contrast, the average score of the Control Group had increased from the game-based testing. A claim could be made that, because we had college (or higher) educated testers, and we largely discussed the game after playing it, the testers still

	Device Dash Group	**Control Group**
1.	Answer 12 multiple choice questions about the Critical Controls from the testers' own knowledge (knowledge-based testing).	
2.	Read the SANS' version of Critical Control 1, 5, and 8 for 10 min, and answer 12 questions (same questions as from 1 but rephrased and re-ordered) on them (reading-based testing).	
3.	Read Device Dash's walkthrough and play the game for 30 min (replay until the 30 min are up). Answer 12 questions on Device Dash (re-phrased and re-ordered questions) (game-based testing).	Play *Meerca Chase* for 30 min (replay until the 30 min are up). Answer 12 questions on *Meerca Chase* (re-phrased and re-ordered questions) (game-based testing).
4.	Answer the same questions from 3 a week later to see if the students retained the information (one-week-later testing).	

TABLE 8.1: Testing Alterations Proposal 1

remembered the answers to the questions after one week. However, these conditions could be checked by re-designing our tests and re-testing Device Dash.

8.2 Re-Designing Our Testing

Our introduction from the reading-based testing was too simple and explained everything that the game teaches. This made it impossible for the testers to learn beyond what we intended them to learn when they played the game. Therefore, instead of giving the testers our own explanation of the 3 Critical Controls, we could have given them the SANS' version in which they would have had to sift through information as well as understand it.

Our testing phase would have been similar to our current testing procedure in that we would have a Device Dash Group and a Control Group. See Table 8.1 for information on the alterations to our testing.

Although the previous approach is a good way to test our game, another testing procedure might have been even better. To better evaluate the difference between learning from reading about the Critical Controls and learning from playing Device Dash, we could have a Reading Group and a Device Dash Group, as laid out in Table 8.2. As a bonus, we could use the results from the reading-based testing carried out for this thesis and would only need to find and test a Device Dash Group as described in Table 8.2.

	Reading Group	**Device Dash Group**
1.	Answer 12 multiple choice questions about the Critical Controls from the testers' own knowledge (knowledge-based testing).	
2.	Read our introduction to the Critical Control 1, 5, and 8 for 10 min, and answer 12 questions (same questions as from 1 but rephrased and re-ordered) on them (reading-based testing).	Read Device Dash's walkthrough and play the game for 30 min (replay until the 30 min are up). Answer 12 questions on Device Dash (rephrased and re-ordered questions from 1) (game-based testing).

TABLE 8.2: Testing Alterations Proposal 2

8.3 Future Work and Optimizations

Although Device Dash has been completed, there are still some improvements that we would like to make in the future. One of them deals with graphics: the current graphics look flat and could be thought of as placeholders for better graphics. For example, the icons used for people are circles. We would replace them with fully-drawn people holding devices. We would also like to add music and sounds that would trigger based on the player's actions.

In addition, we would implement more sections of the Critical Controls in Device Dash, as well as make the concepts more accurate, in terms of what a real admin would do. For example, we could implement the cleaning of compromised devices, which would make it possible to allow devices to stay on the network, once they have been cleaned. Players could select whether or not to scan and clean unauthorized devices and, if cleaned, change their status to authorized and allow them to stay on the network. These authorized devices would act just like any other authorized device.

Another improvement would be to deal with the issue that Critical Control 1 raises: authorized devices cannot be compromised in CC1, although they can be compromised in reality. To assuage this issue, we would make the probability of authorized devices becoming compromised small compared to that of unauthorized devices. In this case, the player would understand that unauthorized devices are being compromised more frequently than authorized devices are, due to their lack of management and increased lack of updates or patches.

Furthermore, we would adjust the scoring to better match the importance of each part of the admin's job. Since removing compromised devices is a very important part of the player's job, devices that become compromised should penalize more points. Currently, we have a linear penalty function that subtracts points from the player, after the player removes a compromised device, based on how long the device has been compromised. We

would replace this function with a steep step function that would teach the importance of removing compromised devices as soon as possible.

There is also a need to convey to the player the trade-off between convenience and security, which admins deal with on a daily basis. Although the ideal would be to have higher security, sometimes a corporation's needs trump the cost imposed by security. For example, Internet browsing can lead to malware infections. However, a company cannot completely block its employees from accessing the Internet or their emails in order to thwart attackers. It can prevent employees from looking at certain websites that can potentially contain infections, taking a less strict approach. We would incorporate these trade-offs in Device Dash by allowing players to make decisions and show the repercussions of those decisions. *CyberCIEGE*, a game discussed in Chapter 3, does this by having the employees of the player's company tell the player how they feel about his decisions after these choices have been made. We could use a similar approach: by incorporating the passage of time, we could show in graphs, charts, and images how the employees are feeling about the security rules being implemented after each month. This would not only make the game more interesting and entertaining to play, but would also have the potential of better expressing the goals and tribulations of being a system administrator.

Appendix A

Testing Introduction

The following introduction was given to the testers to read before playing Device Dash, during the Reading-Based Testing.

Intro:

Cyber attacks have greatly increased over the last few years and preventing them has become a high priority for our nation. Automated defenses are required to keep up with the high volume of attacks. Recently, 20 Critical Security Controls (CCs) have been developed to mitigate the most important types of attacks. We focus on preventing the three attack steps illustrated in Figure A.1.

FIGURE A.1: Compromization From Unauthorized Device

In the first step, an attacker from the Internet scans your network, looking for an unauthorized device in it, such as the iPod shown in this picture. These unauthorized devices are assumed to be unsecure and can be easily compromised by the attacker. Once compromised, the attacker can gain a foothold into your network. In the second step, the attacker searches inside the network for other valuable systems and attempts to break in. He compromises a system administrators computer and steals the credentials used by the administrator to manage other computers on the network. As a result, any computer that is accessible to the system administrator is also accessible to the attacker. In this final step, the attacker has gained full control of your network.

Three controls, or preventive, measures have been proposed to prevent one or more of these attack steps:

- Critical Control 1: Inventory of Authorized and Unauthorized Devices
- Critical Control 5: Boundary Defense
- Critical Control 8: Controlled Use of Administrative Privileges

The following sections will describe in detail the attacker tactics that motivate these controls and the desired defender behavior for each of them.

Inventory of Authorized and Unauthorized Devices Attackers scan the address spaces of organizations, continuously, looking for unauthorized devices and/or devices that have not been checked and cleaned from malware. These devices are the most vulnerable and, therefore, the easiest to attack due to their potential lack of installed security updates or patches. Furthermore, once the attackers gain an entry to a device, they can use that device as an anchor point to break into another device or system, which might not be penetrable through the previously mentioned exploit.

The need for an automated and continuous mechanism to detect such attacker scans or backdoors is, therefore, evident. In the real world, this is done through the use of an asset inventory, where each device's information (MAC address, IP address, device type, device name, owner, etc.) is listed, and unauthorized devices' behaviors are continuously monitored for unusual or suspicious behavior through the use of a scanner. (This could also be done through continuously monitoring the system by hand, but would be highly inefficient.) Network Access Control (NAC) could also be implemented. NAC allows only devices that have met the necessary requirements, which include being up to date with security patches, having updated antivirus software, etc., to connect to the company's network.

Boundary Defense

It is often easier to attack from within a network than it is to attack from the outside because there are, usually, fewer internetworking restrictions. So, attackers could gain full access to a computer on the network and use it as a pivoting point to get access to other computers/devices connected to the same network. This would ensure the spreading of infections and the compromise of many computers, in order to get access to all of their data or attack the corporation.

To thwart this, the network could be subdivided into multiple subnetworks (subnets). Access from one subnet to another should be restricted such that, even if an attacker successfully penetrates one subnet, the damage is limited.

Controlled Use of Admin Privileges Another attacker plan consists of compromising administrators. Admin users have abilities and privileges that regular users do not, such as installing software, changing (user, program, device, etc.) permissions, etc. But, this also means that admins have to be more careful than other users, since their actions are more dangerous: downloading and opening a questionable attachment could mean infecting the whole computer and all of its users, which could result in the attackers' gaining access to all of that computer's sensitive data.

Administrator systems should be monitored more closely. Automated and continuous defense systems, such as address space scanners or NAC, should be used, and activities on these systems should be carefully audited.

Appendix B

Testing Questions

The testers were asked to answer the following questions after each testing phase.

Era Vuksani Game Questions – Knowledge-Based Test Questions

General questions:

1. Name:
2. Age:
3. Gender: Male Female
4. Do you currently play any videogame(s) on a regular basis? If so, which videogame(s):

5. What style of videogames do you like/play (circle all that pertain): Shoot-em-ups, RPG/MMORPG, Action, Adventure, Puzzles

Other:_____

6. Are you a CS major? Yes No
7. Do you know a lot about CS? Yes A little No

****** Answer each of the questions below based on your own personal knowledge. ******

Each of the questions below contain a "Confidence rating" portion on the right-hand side. Please fill it in with a number from 1 – 5 (1 being the least confident, 5 being the most confident).

1. What does Network Access Control mainly achieve? Confidence rating: ___
   A. Allows unauthorized users to automatically join a corporate network
   B. Prevents unauthorized users from participating on a corporate network
   C. Keeps a record of unauthorized access to a corporate network
   D. Scans for malware on the network

2. A user is most likely to become compromised: Confidence rating: ___
   A. After he/she has been authorized by the system administrator to join the network
   B. Before he/she has been authorized by the system administrator to join the network
   C. After he/she has joined the network without authorization from the system administrator
   D. While he/she is waiting for authorization from the system administrator to join the network

3. What is more important (circle all that pertain): Confidence rating: ___
   A. Removing unauthorized devices
   B. Allowing authorized devices
   C. Supervising employees that are online
   D. Knowing where employees are at all times

4. When can devices become compromised: Confidence rating: ___
   A. When they are authorized
   B. When they are unauthorized and are waiting to be authorized
   C. When they are removed from the network
   D. When they are unauthorized and are in the address space

5. Why is it important to subdivide a network: Confidence rating: ___
   A. Because, by dividing it, we can make sure that, even if part of the network is compromised (or infected), the other parts of the network are not
   B. Because, by dividing it, we can make sure that detection will occur more rapidly, since

the scanner now needs to scan a smaller region of the network to find the compromised devices and remove them.

C. Because, by dividing it, we don't really need firewalls anymore, since we have the divisions
D. This is a trick question: it is not important to subdivide a network

6. What is the most effective protection method for a company: Confidence rating: ___

A. Update your system once a year
B. Install an automated and continuous defense system
C. Have admins manually check the network for malware
D. Never run a system as an admin

7. What can admin devices do that is extremely harmful: Confidence rating: ___

A. Remove devices from the network
B. Compromise (or infect) other devices, if they, themselves, have been compromised
C. Allow you to install anti-virus software
D. Scan devices for malware

8. Why is it important to scan your network for unauthorized devices?
Confidence rating: ___

A. To make sure that passwords are changed often
B. To make sure that people are using up-to-date software, which has no holes/bugs
C. To know if there are any unauthorized devices on the network and remove them, if the need arises
D. So that you can find out which device (if any) is attacking other devices

9. Why is it dangerous to connect an unauthorized device to the company's network?
Confidence rating: ___

A. Unauthorized devices take up more address spaces on the network than authorized devices do
B. Unauthorized devices are more prone to being compromised (or infected)
C. Unauthorized devices are more receptive to looking at what other devices are doing
D. Unauthorized devices are just as dangerous as authorized devices

10. Once they get a foothold in the company (ie: have infected a computer with malware and have full access to it), attackers, typically, tend to attack: Confidence rating: ___

A. A computer that is authorized to be on the network
B. A device on the outside of the company, because it's easier
C. A computer that is up to date
D. A device on the inside of the company, because it's easier

11. If you have limited resources, what should you do to contain the spread of infections?
Confidence rating: ___

A. Separate the network into subnetworks
B. Scan your address space to look for unauthorized devices
C. There is no known way to contain the spread of infections
D. Update all devices, on the network, constantly

12. What increases the chances of devices getting compromised? Confidence rating: ___
    A. The shorter amount of time that an unauthorized device is on the network, the more chance there is that that device will become compromised
    B. The shorter amount of time that an authorized device is on the network, the more chance there is that that device will become compromised
    C. The longer an unauthorized device is on the network, the more chance there is that that device will become compromised
    D. The longer an authorized device is on the network, the more chance there is that that device will become compromised

Reading-Based Questions

Name:

****** Answer each of the questions below based on the reading, not from practice or your own personal knowledge. ******

Each of the questions below contain a "Confidence rating" portion on the right-hand side. Please fill it in with a number from 1 – 5 (1 being the least confident, 5 being the most confident).

1. Administrative devices can cause harm: Confidence rating: ___
   A. By installing anti-virus programs for users that don't want them
   B. By compromising devices on the network, once the admin device has been compromised
   C. By looking at each device, on the network, for malware
   D. By removing all (authorized and unauthorized) devices from the network

2. Often, a computer will get compromised when: Confidence rating: ___
   A. If, without authorization, it joins the network
   B. If, with authorization, it joins the network
   C. If it has joined the network and is waiting for authorization from the system administrator
   D. Before authorization by the system administrator has been given

3. The main purpose of Network Access Control (NAC) is to: Confidence rating: ___
   A. Prevent authorized users from automatically joining the network
   B. Track all unauthorized network access
   C. Check the network for malware
   D. Only allow authorized users to join the network

4. Subdividing a network is important because: Confidence rating: ___
   A. Malware detection occurs quicker, when the network is subdivided, since the scanners are then able to scan a smaller region of the network
   B. It's important to scan a network for malware, but not to subdivide it
   C. At that point, we don't need firewalls anymore, since the subdivisions act like firewalls
   D. Even if part of the network is compromised, by dividing the network, we can make sure that the other parts of the network are not

5. Circle the most important to a system administrator (circle all that pertain):
   Confidence rating: ___
   A. Keeping track of employee locations
   B. Permitting authorized devices
   C. Knowing what employees are doing, if they are online
   D. Taking unauthorized devices off of the network

6. Scans that check for authorized and unauthorized devices should be performed because:
   Confidence rating: ___
   A. It is important to know which devices have been infiltrated by attackers
   B. It is important to know if there are devices that are unauthorized and, if need be, remove them from the network
   C. It is important to check that passwords are changed often
   D. It is important to ensure that devices use software that is often updated

Reading-Based Questions

7. _____ increases the chances of devices getting compromised. Confidence rating: ___
   A. An increase in the amount of time that an authorized device is on the network
   B. A decrease in the amount of time that an authorized device is on the network
   C. An increase in the amount of time that an unauthorized device is on the network
   D. A decrease in the amount of time that an unauthorized device is on the network

8. Devices may become compromised when: Confidence rating: ___
   A. They are taken off of the network, since that is when they are changed from authorized to unauthorized
   B. They are authorized, since that is when they can be used to infiltrate other devices on the network
   C. They are in the address space, but are unauthorized, since this vulnerability gives attackers an easier entry to them
   D. They are waiting to be authorized, but are unauthorized, since their waiting makes them more vulnerable than if they were online

9. Attackers, once they have attacked a device, on the network of a company, and have administrative permissions on it, love to: Confidence rating: ___
   A. Be lazy: they pick the easiest target to attack, which is a device on the inside
   B. Be lazy: they pick the easiest way to attack a device, which is an authorized computer that is on the network
   C. Be lazy: they pick the easiest way to attack a device, which is an up-to-date computer
   D. Be lazy: they pick the easiest way to attack a device, which is a device on the outside

10. To contain the spread of infections, when resources are limited:
Confidence rating: ___
   A. You should update all device constantly
   B. You should divide the network in to subnets
   C. You should do nothing, since there are no known way to contain the spread of an infection
   D. You should scan your address space constantly to find out if there any unauthorized devices

11. For a company, the most effective protection method is to: Confidence rating: ___
   A. Put into place a continuous and automated protection system
   B. Update your system once a year
   C. Check the system for malware, manually, just to be sure
   D. Never use administrative privileges to run the system

12. You should not connect unauthorized devices to the company's network because:
Confidence rating: ___
   A. Unauthorized devices can look at what other devices are doing
   B. Unauthorized devices require more space to be on the network, than authorized devices do
   C. Authorized devices and unauthorized devices are equally dangerous
   D. It is easier to compromise unauthorized devices

Game-Based Questions

Name:

****** Answer each of the questions below based on the game that you just played, not from practice or your own personal knowledge. ******

Each of the questions contain a "Confidence rating" portion on the right-hand side. Please fill it in with a number from 1 – 5 (1 being the least confident, 5 being the most confident).

1. The main reason that subdivisions in a network are important is: Confidence rating: ___
   A. Subdivisions are actually not important, but it is important to scan a network for malware
   B. Because by dividing a network, we can minimize the spread of infections
   C. Because they eliminate the need for firewalls
   D. Because scans have to scan through a smaller section of the network to find malware

2. ____ to contain the spread of infections, when you have limited resource. Confidence rating: ___
   A. There is no known way
   B. Use walls to subdivide your network into subnets
   C. Update all devices on the network, constantly,
   D. Scan your address space, looking for unauthorized device,

3. Network Access Control (NAC) is mostly useful because: Confidence rating: ___
   E. It tracks the open processes by unauthorized devices
   F. It makes sure that authorized users do not automatically join the network
   G. It scans the network for malware
   H. It keeps unauthorized users from joining the network

4. Attackers pick the easiest targets, once they have full access to a computer in a company, which are: Confidence rating: ___
   A. Up-to-date devices on the network
   B. Authorized devices on the network
   C. Devices on the outside
   D. Devices on the inside

5. The best defense against infections is to: Confidence rating: ___
   A. Simply not install programs as an admin
   B. Manually check for malware or infections
   C. Scan the address space of the network continuously and automatically
   D. Update your computers and programs once a year

6. Devices running under administrators are harmful: Confidence rating: ___
   A. Because they can install anti-virus software without other users' permission
   B. Because they can potentially compromise all other devices on the network, once they have been compromised
   C. Because they can look for malware on the network without the permission of the other users or devices
   D. Because they can potentially remove any and all devices from the network without permission

Game-Based Questions

7. A device may get compromised when: Confidence rating: ___
   A. It has not yet been authorized by the admin
   B. It has been authorized and joins the network
   C. It is waiting for authorization from the system administrator, but has already joined the network
   D. It joins the network without authorization

8. Network scans need to be performed frequently to: Confidence rating: ___
   A. Know if there are unauthorized devices on the network and remove them.
   B. Know which devices have been attacked by attackers
   C. Know which devices are up-to-date and which are not
   D. Know if passwords on those devices have been changed or not

9. System admins care the most about (circle all that pertain): Confidence rating: ___
   A. Making sure that authorized devices can join the network
   B. Keeping track of what employees are doing, if they are online
   C. Keeping track of an employee current location
   D. Making sure that unauthorized devices stay off of the network

10. Unauthorized devices should not be connected to the network because:
Confidence rating: ___
   A. When on the network, unauthorized devices require more memory than authorized devices
   B. Unauthorized devices have the ability to see what other devices are up to
   C. Both unauthorized and authorized devices should not be connected to the network
   D. Unauthorized devices are the easiest to compromise

11. Devices become compromised: Confidence rating: ___
   A. When they are unauthorized, but are in the address space, since attackers can compromise them more easily, at this time
   B. Just as they are taken off of the network, when their status changes from authorized to unauthorized
   C. When they are waiting to get authorized, since their waiting makes them more vulnerable than if they were online
   D. When they become authorized and can attack other devices on the network

12. ____ devices have a greater chance to be compromised, when ____.
Confidence rating: ___
   A. Authorized; they spend more time on the network
   B. Authorized; they spend less time on the network
   C. Unauthorized; they spend more time on the network
   D. Unauthorized; they spend less time on the network

Appendix C

Game Items

Appendix C contains the items that were available in the game.

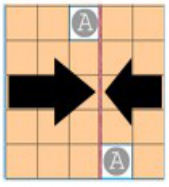
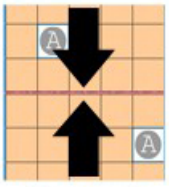
Item	Description	Cost	Required Level To Buy
 Timegain	Slows down the arrival rate of the devices for 5 seconds.	10 pts	Level 2+
 Bronze Scanner (slow)	Scans one IP address per sec and displays all unauthorized devices detected.	50 pts	Level 3+
 Silver Scanner (fast)	Scans one IP address per 0.5sec and displays all unauthorized devices detected.	100 pts	Level 5+
 Golden Scanner (NAC)	Automatically monitors devices, removing unauthorized devices, trying to get on the network.	150 pts	Level 7+
 Vertical Wall	Split your address space into two sections, making it imposible for viruses to spread from one section to the other.	20 pts	Level 4+
 Horizontal Wall	Split your address space into two sections, making it impossible for viruses to spread from one section to the other.	40 pts	Level 6+

FIGURE C.1: Items, Descriptions and Requirements

Appendix D

Full Testing Results

Appendix D contains a detailed version of the results for each testing phase, starting with the Knowledge-Test results.

		Group 1						
		Apr 2nd			Apr 3rd	Apr 4th	Apr 6th	
Question #		#1	#2	#3	#4	#5	#6	#7
	Score:	0	1	1	1	0	1	1
1	Confidence:	0	1	4	4	0	5	5
	Score:	0	0	1	0	0	0	1
2	Confidence:	2	0	2	2	0	5	2
	Score:	0.5	0	0.5	1	0.5	1	1
3	Confidence:	3	0	2	3	1	3	3
	Score:	1	0	0	0	0	0	1
4	Confidence:	2	0	0	4	0	5	3
	Score:	1	1	1	1	1	1	1
5	Confidence:	3	1	2	4	3	5	5
	Score:	1	1	1	1	1	1	1
6	Confidence:	3	1	3	3	4	5	3
	Score:	1	1	1	1	1	1	1
7	Confidence:	5	1	3	5	3	5	5
	Score:	1	1	0	1	1	1	1
8	Confidence:	4	1	2	3	1	5	4
	Score:	1	0	1	1	0	1	1
9	Confidence:	5	0	3	4	0	3	5
	Score:	1	0	0	1	0	0	1
10	Confidence:	3	1	3	3	0	5	3
	Score:	0	0	0	1	0	1	1
11	Confidence:	2	0	1	3	0	3	3
	Score:	1	0	1	1	0	0	1
12	Confidence:	5	0	3	4	0	1	4
Overall Score (/12):		8.5	5	7.5	10	4.5	8	12
Overall Confidence (/60):		37	6	28	42	12	50	45

Group 2					
Apr 2nd		Apr 3rd	Apr 4th		
#8	#9	#10	#11	#12	#13
1	1	1	0	1	1
3	4	3	0	2	4
1	1	0	0	0	1
2	3	0	0	0	2
1	0.5	1	0.5	1	0.5
3	4	4	2	3	4
1	0	0	0	0	1
2	3	0	0	2	3
1	0	0	0	0	1
4	3	0	0	0	3
1	1	0	0	0	0
2	5	0	0	2	2
1	1	0	1	0	1
4	4	2	3	0	4
1	1	0	0	1	1
4	3	2	2	4	4
1	1	1	1	1	0
4	4	5	2	3	4
1	0	0	0	0	0
4	3	0	0	0	4
1	0	0	0	0	0
3	4	2	0	2	3
1	1	1	1	1	1
5	4	3	2	3	2
12	7.5	4	3.5	5	7.5
40	44	21	11	21	39

Avg. score G1:	7.93
Avg. confidence G1:	31.43
Avg. score G2:	6.58
Avg. confidence G2:	29.33

G1 and G2 percent differences	
score:	11.21
conf:	3.49

FIGURE D.1: Knowledge Test - Results

		Group 1						
		Apr 2nd			Apr 3rd	Apr 4th	Apr 6th	
Question #		**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**
	Score:	1	1	1	1	0	1	0
3	Confidence:	3	5	5	4	3	5	4
	Score:	1	1	1	1	1	1	1
2	Confidence:	4	3	4	5	2	5	4
	Score:	1	1	0.5	1	0.5	0	1
5	Confidence:	4	1	3	5	1	4	4
	Score:	1	1	1	1	0	0	1
8	Confidence:	5	4	4	5	0	4	4
	Score:	1	1	1	1	1	1	1
4	Confidence:	5	5	4	5	5	5	4
	Score:	1	1	1	1	1	1	0
11	Confidence:	5	1	4	5	4	5	4
	Score:	1	1	1	1	1	1	1
1	Confidence:	5	5	4	5	4	5	5
	Score:	1	1	0	1	1	1	1
6	Confidence:	4	3	3	5	2	4	5
	Score:	1	1	1	1	1	1	1
12	Confidence:	5	5	4	5	3	5	5
	Score:	1	0	1	0	0	1	0
9	Confidence:	5	2	4	5	2	5	3
	Score:	1	1	0	1	1	1	1
10	Confidence:	4	2	4	5	4	4	4
	Score:	1	1	1	1	1	1	1
7	Confidence:	5	5	4	5	4	5	5
Overall Score:		12	11	9.5	11	8.5	10	9
Score change:		3.5	6	2	1	4	2	-3
Overall Confidence:		54	41	47	59	34	56	51
Confidence change:		17	35	19	17	22	6	6

Group 2					
Apr 2nd		Apr 3rd	Apr 4th		
#8	**#9**	**#10**	**#11**	**#12**	**#13**
1	0	0	0	0	1
5	3	3	0	4	4
1	0	1	0	0	1
5	3	4	2	0	5
1	0.5	1	0	1	1
5	3	3	0	4	4
1	1	1	1	1	1
5	3	3	1	3	4
1	0	1	1	1	1
5	3	5	4	5	5
1	1	1	1	0	1
5	4	4	4	3	4
1	1	1	1	1	1
5	3	5	4	5	5
1	0	1	0	0	1
5	2	4	3	4	3
1	1	1	1	1	1
5	4	3	2	5	3
1	1	1	0	0	1
5	4	3	4	4	5
1	0	1	1	1	0
5	4	4	3	5	4
1	1	1	0	1	1
5	3	4	1	4	5
12	6.5	11	6	7	11
0	-1	7	2.5	2	3.5
60	39	45	28	46	51
20	-5	24	17	25	12

	current	change	% change
Avg. score G1:	10.14	2.21	18.45
Avg. confidence G1:	48.86	17.43	29.05
Avg. score G2:	8.92	2.33	19.44
Avg. confidence G2:	44.83	15.50	25.83

G1/G2 % diff	
score	10.22
conf	6.71

FIGURE D.2: Reading Test - Results (Note: Questions are ordered so as to match those from Figure D.1.)

		Group 1						
Question #		#1	#2	#3	#4	#5	#6	#7
	Score change	1	0	0	0	0	0	-1
3	Confidence ch	3	4	1	0	3	0	-1
	Score change	1	1	0	1	1	1	0
2	Confidence ch	2	3	2	3	2	0	2
	Score change	0.5	1	0	0	0	-1	0
5	Confidence ch	1	1	1	2	0	1	1
	Score change	0	1	1	1	0	0	0
8	Confidence ch	3	4	4	1	0	-1	1
	Score change	0	0	0	0	0	0	0
4	Confidence ch	2	4	2	1	2	0	-1
	Score change	0	0	0	0	0	0	-1
11	Confidence ch	2	0	1	2	0	0	1
	Score change	0	0	0	0	0	0	0
1	Confidence ch	0	4	1	0	1	0	0
	Score change	0	0	0	0	0	0	0
6	Confidence ch	0	2	1	2	1	-1	1
	Score change	0	1	0	0	1	0	0
12	Confidence ch	0	5	1	1	3	2	0
	Score change	0	0	1	-1	0	1	-1
9	Confidence ch	2	1	1	2	2	0	0
	Score change	1	1	0	0	1	0	0
10	Confidence ch	2	2	3	2	4	1	1
	Score change	0	1	0	0	1	1	0
7	Confidence ch	0	5	1	1	4	4	1

Group 2					
#8	#9	#10	#11	#12	#13
0	-1	-1	0	-1	0
2	-1	0	0	2	0
0	-1	1	0	0	0
3	0	4	2	0	3
0	0	0	-0.5	0	0.5
2	-1	-1	-2	1	0
0	1	1	1	1	0
3	0	3	1	1	1
0	0	1	1	1	0
1	0	5	4	5	2
0	0	1	1	0	1
3	-1	4	4	1	2
0	0	1	0	1	0
1	-1	3	1	5	1
0	-1	1	0	-1	0
1	-1	2	1	0	-1
0	0	0	0	0	1
1	0	-2	0	2	-1
0	1	1	0	0	1
1	1	3	4	4	1
0	0	1	1	1	0
2	0	2	3	3	1
0	0	0	-1	0	0
0	-1	1	-1	1	3

negative score change	
negative confidence change	

G1	
Total score negatives:	5
Total confidence negatives:	4
G2	
Total score negatives:	8
Total confidence negatives:	12

FIGURE D.3: Reading Test - Results (continued)

		Group 1						
		Apr 2nd			Apr 3rd	Apr 4th	Apr 6th	
Question #		**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**
	Score:	1	1	1	1	1	1	1
3	Confidence:	4	5	5	4	3	5	5
	Score:	1	1	1	1	1	0	1
7	Confidence:	5	5	4	5	2	5	4
	Score:	1	1	0.5	1	1	0.5	1
9	Confidence:	5	3	4	5	3	5	4
	Score:	1	1	1	1	0	1	1
11	Confidence:	5	4	5	5	1	5	5
	Score:	1	1	1	1	1	1	1
1	Confidence:	5	5	5	5	4	5	5
	Score:	1	1	1	1	1	1	0
5	Confidence:	5	3	5	5	4	5	4
	Score:	1	1	1	1	1	1	1
6	Confidence:	5	4	5	5	5	5	5
	Score:	1	1	1	1	1	1	1
8	Confidence:	4	5	5	5	2	5	5
	Score:	1	1	1	1	1	1	1
10	Confidence:	5	5	5	5	5	5	5
	Score:	1	0	1	1	0	1	1
4	Confidence:	5	3	4	5	1	5	3
	Score:	1	1	0	1	1	1	1
2	Confidence:	4	4	4	5	4	5	4
	Score:	1	1	1	1	1	1	1
12	Confidence:	5	5	5	5	2	5	5
Overall Score:		12	11	10.5	12	10	10.5	11
Score change:		0	0	1	1	1.5	0.5	2
Overall Confidence:		57	51	56	59	36	60	54
Confidence change:		3	10	9	0	2	4	3

	Group 2					
	Apr 2nd		Apr 3rd	Apr 4th		
	#8	**#9**	**#10**	**#11**	**#12**	**#13**
Score:	1	1	1	1	0	1
Confidence:	5	4	4	1	4	4
Score:	1	1	1	0	0	1
Confidence:	5	3	4	2	3	4
Score:	1	0.5	1	1	1	0.5
Confidence:	5	4	4	2	5	4
Score:	1	1	1	1	0	1
Confidence:	5	4	3	2	0	4
Score:	1	0	1	1	1	1
Confidence:	5	4	4	5	5	5
Score:	1	1	1	0	0	0
Confidence:	5	3	4	1	3	5
Score:	1	1	1	1	1	1
Confidence:	5	4	4	4	5	5
Score:	1	1	1	0	0	1
Confidence:	5	3	4	1	4	5
Score:	1	1	1	1	1	1
Confidence:	5	4	3	2	5	5
Score:	1	1	0	1	0	1
Confidence:	5	5	3	5	5	5
Score:	1	0	1	1	1	0
Confidence:	5	3	3	2	4	4
Score:	1	1	1	1	1	1
Confidence:	5	4	4	2	5	5
	12	9.5	11	9	6	9.5
	0	3	0	3	-1	-1.5
	60	45	44	29	48	55
	0	6	-1	1	2	4

	current	change	% change
Avg. score G1:	11.00	0.86	7.14
Avg. confidence G1:	53.29	4.43	7.38
Avg. score G2:	9.50	0.58	4.86
Avg. confidence G2:	46.83	2.00	3.33

G1/G2 % diff	
score	12.50
conf	10.75

FIGURE D.4: Game Test - Results

		Group 1						
Question #		**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**
	Score change:	0	0	0	0	1	0	1
3	Confidence ch:	1	0	0	0	0	0	1
	Score change:	0	0	0	0	0	-1	0
2	Confidence ch:	1	2	0	0	0	0	0
	Score change:	0	0	0	0	0.5	0.5	0
5	Confidence ch:	1	2	1	0	2	1	0
	Score change:	0	0	0	0	0	1	0
8	Confidence ch:	0	0	1	0	1	1	1
	Score change:	0	0	0	0	0	0	0
4	Confidence ch:	0	0	1	0	-1	0	1
	Score change:	0	0	0	0	0	0	0
11	Confidence ch:	0	2	1	0	0	0	0
	Score change:	0	0	0	0	0	0	0
1	Confidence ch:	0	-1	1	0	1	0	0
	Score change:	0	0	1	0	0	0	0
6	Confidence ch:	0	2	2	0	0	1	0
	Score change:	0	0	0	0	0	0	0
12	Confidence ch:	0	0	1	0	2	0	0
	Score change:	0	0	0	1	0	0	1
9	Confidence ch:	0	1	0	0	-1	0	0
	Score change:	0	0	0	0	0	0	0
10	Confidence ch:	0	2	0	0	0	1	0
	Score change:	0	0	0	0	0	0	0
7	Confidence ch:	0	0	1	0	-2	0	0

Group 2					
#8	**#9**	**#10**	**#11**	**#12**	**#13**
0	1	1	1	0	0
0	1	1	1	0	0
0	1	0	0	0	0
0	0	0	0	3	-1
0	0	0	1	0	-0.5
0	1	1	2	1	0
0	0	0	0	-1	0
0	1	0	1	-3	0
0	0	0	0	0	0
0	1	-1	1	0	0
0	0	0	-1	0	-1
0	-1	0	-3	0	1
0	0	0	0	0	0
0	1	-1	0	0	0
0	1	0	0	0	0
0	1	0	-2	0	2
0	0	0	0	0	0
0	0	0	0	0	2
0	0	-1	1	0	0
0	1	0	1	1	0
0	0	0	0	0	0
0	-1	-1	-1	-1	0
0	0	0	1	0	0
0	1	0	1	1	0

negative score change
negative confidence change

G1	
Total score negatives:	1
Total confidence negatives:	4

G2	
Total score negatives:	5
Total confidence negatives:	11

FIGURE D.5: Game Test - Results (continued)

		Group 1						
		Apr 9th			Apr 10th	Apr 11th	Apr 13th	
Question #		**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**
	Score:	1	1	1	1	1	1	
3	Confidence	4	5	5	5	4	5	
	Score:	1	1	1	1	1	1	
7	Confidence	5	4	5	5	2	5	
	Score:	1	1	1	1	0.5	1	
9	Confidence	4	2	4	5	4	5	
	Score:	1	1	1	1	0	1	
11	Confidence	5	5	5	5	0	5	
	Score:	1	1	1	1	1	1	
1	Confidence	5	5	5	5	4	5	
	Score:	1	1	1	1	1	1	
5	Confidence	5	2	5	5	5	5	
	Score:	1	1	1	1	1	1	
6	Confidence	5	4	5	5	5	5	
	Score:	1	1	1	1	1	1	
8	Confidence	4	5	4	5	3	5	
	Score:	1	1	1	1	1	1	
10	Confidence	5	5	5	5	5	5	
	Score:	1	0	0	1	1	1	
4	Confidence	5	2	3	5	1	5	
	Score:	1	0	0	1	1	1	
2	Confidence	5	3	5	5	4	5	
	Score:	1	1	1	1	1	1	
12	Confidence	5	5	5	5	1	5	
Overall Score:		12	10	10	12	10.5	12	
Score change:		0	-1	-0.5	0	0.5	1.5	
Overall Confidence:		57	47	56	60	38	60	
Confidence change:		0	-4	0	1	2	0	

	Group 2					
Apr 9nd		Apr 10rd	Apr 11th			
#8	**#9**	**#10**	**#11**	**#12**	**#13**
1	0	1	1	1	0
5	5	4	2	5	4
1	1	1	1	0	1
5	4	4	1	4	4
1	1	0.5	0.5	1	1
5	3	4	3	5	3
1	1	1	1	1	1
5	3	3	3	3	4
1	0	1	1	1	1
5	4	4	4	5	4
1	1	1	0	0	1
5	2	4	2	4	5
1	1	1	1	1	1
5	5	4	3	5	5
1	0	1	1	0	1
5	3	5	1	3	4
1	1	1	1	1	1
5	5	4	1	5	5
1	1	0	1	0	1
5	4	3	3	5	5
1	0	1	1	1	0
5	3	4	3	5	3
1	1	1	1	1	1
5	4	4	1	5	5
12	8	10.5	10.5	8	10
0	-1.5	-0.5	1.5	2	0.5
60	45	47	27	54	51
0	0	3	-2	6	-4

	current	change	% change
Avg. score G1:	11.08	0.08	0.69
Avg. confidence G1:	53.00	-0.17	-0.28
Avg. score G2:	9.83	0.33	2.78
Avg. confidence G2:	47.33	0.50	0.83

G1/G2 % diff	
score	10.42
conf	0.09

FIGURE D.6: One Week Later Test - Results

		Group 1						
Question #		**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**
	Score char	0	0	0	0	0	0	
3	Confidence	0	0	0	1	1	0	
	Score char	0	0	0	0	0	1	
7	Confidence	0	-1	1	0	0	0	
	Score char	0	0	0.5	0	-0.5	0.5	
9	Confidence	-1	-1	0	0	1	0	
	Score char	0	0	0	0	0	0	
11	Confidence	0	1	0	0	-1	0	
	Score char	0	0	0	0	0	0	
1	Confidence	0	0	0	0	0	0	
	Score char	0	0	0	0	0	0	
5	Confidence	0	-1	0	0	1	0	
	Score char	0	0	0	0	0	0	
6	Confidence	0	0	0	0	0	0	
	Score char	0	0	0	0	0	0	
8	Confidence	0	0	-1	0	1	0	
	Score char	0	0	0	0	0	0	
10	Confidence	0	0	0	0	0	0	
	Score char	0	0	-1	0	1	0	
4	Confidence	0	-1	-1	0	0	0	
	Score char	0	-1	0	0	0	0	
2	Confidence	1	-1	1	0	0	0	
	Score char	0	0	0	0	0	0	
12	Confidence	0	0	0	0	-1	0	

Group 2					
#8	**#9**	**#10**	**#11**	**#12**	**#13**
0	-1	0	0	1	-1
0	1	0	1	1	0
0	0	0	1	0	0
0	1	0	-1	1	0
0	0.5	-0.5	-0.5	0	0.5
0	-1	0	1	0	-1
0	0	0	0	1	0
0	-1	0	1	3	0
0	0	0	0	0	0
0	0	0	-1	0	-1
0	0	0	0	0	1
0	-1	0	1	1	0
0	0	0	0	0	0
0	1	0	-1	0	0
0	-1	0	1	0	0
0	0	1	0	-1	-1
0	0	0	0	0	0
0	1	1	-1	0	0
0	0	0	0	0	0
0	-1	0	-2	0	0
0	0	0	0	0	0
0	0	1	1	1	-1
0	0	0	0	0	0
0	0	0	-1	0	0

negative score change	
negative confidence cha	

G1	
Total score negatives:	3
Total confidence negati	10
G2	
Total score negatives:	5
Total confidence negati	15

FIGURE D.7: One Week Later Test - Results (continued)

Appendix E

Device Dash Code

Appendix E contains all of the code that was used in the game.

Main.as

```
package
{
        import flash.display.*;
        import flash.events.*;
        import flash.sampler.NewObjectSample;
        import flash.utils.getTimer;
        import flash.utils.Timer;
        import flash.geom.*;
        import flash.text.*;

//*remove?:
        import fl.transitions.*;

        import flash.ui.Keyboard;

        import classes.Box;
        import classes.Person;
        //import classes.traceDL;
        import classes.BoxTop;
        import classes.PointBurst;
        import classes.ItemsPanel;
        import classes.GameData;
        import classes.SepOutline;
        import classes.Messages;


//!remove
        import NextLevel;
```

```
  /**
   * Main class for Critical Control 1 Metrics Game.
   * @author Era Vuksani
   */
  //[Frame(factoryClass = "Preloader")]
        public class Main extends MovieClip {
  //contains the items in the game, excluding the box tops
  private static var container:Sprite = new Sprite();
  //contains all of the people objects
  private static var peopleContainer:Sprite = new Sprite();
  //contains the box tops
  private static var boxTopsContainer:Sprite = new Sprite();
  //contains game over screen, etc..
  private static var miscContainer:Sprite = new Sprite();

  //objects:
  private static var form:Form = new Form(); //form that the person needs to fill
  out
  private static var table:Table = new Table();//desk where player sits
  private static var lineSeg:MovieClip = new MovieClip(); //line on left-hand
  side of address boxes
  private static var hourglass:Hourglass = new Hourglass();//hourglass object
  that contains num of hourglasses in inv
  private static var hourglassMouseOver:HourglassMouseOver = new
  HourglassMouseOver();//mouse over text


  //timer objects:
  //time form takes to get from desk to person
  private static var timeForm:Timer = new Timer(100, 20);//!!!5, 1);
  private static var myTimer:Timer; //delays the movement of each person on the
  line
  //figures out when the person in each box will turn to another color
  //private static var greyToRedTimer:Timer;
  private static var deltaMood:int; //to keep track of the elapsed time between
  when the object was
  //created and the next mood stage
  //private var timeWait:Timer;//time to wait between moving object toward area3
  and getting
  //authorized
  private static var timePersonCreation:Timer;//creation of new person object
  private static var pauseTimer:int; //accounts for time game has been paused
  //time allocation objects
  private static var deltaTime:Number = 10;// 5; //time it takes unauthorized
  circles to get to boxes

  //object arrays
//?  private static var fixedObjects:Array = new Array(); //address locations,
floor, ceiling, walls
  private static var addresses:Array = new Array(); //addresses for the people to
  go to
  private static var tops:Array = new Array(); //contains box tops

  private static var people:Array = new Array(); //list of people/devices
  private static var area3:Array = new Array(); //all of the devices that are in
```

```
the boxes
//list of people in Area 2 (not in line and not in box)
private static var area2:Array = new Array();

//game screens
private static var gameOverSplash:GameOverScreen = new GameOverScreen();//game
over screen
private static var gameOverContainer:Sprite = new Sprite();
//level complete screen
private static var levelCompleteSplash:LevelCompleteScreen = new
LevelCompleteScreen();
private static var levelCompleteContainer:Sprite = new Sprite();
//winner screen
private static var gameCompleteSplash:YouWin = new YouWin();
private static var gameCompleteContainer:Sprite = new Sprite();

//textfields
//text at top left of screen
private static var pointsText:TextField = new TextField();
private static var pointsTextFormat:TextFormat = new TextFormat();//formatting

//text at top right of screen (current level)
private static var levelText:TextField = new TextField();
private static var levelTextFormat:TextFormat = new TextFormat();//formatting

//textfield for the game over screen
private static var pointsTextScreen:TextField = new TextField();
private static var pointsTextScreenFormat:TextFormat = new
TextFormat();//formatting

//textfield for paused game
private static var gamePauseText:GamePauseText = new GamePauseText();

//textfield for the hourglass object
private static var hourglassInventoryText:TextField = new TextField();
private static var hourglassInventoryTextFormat:TextFormat = new TextFormat();

//timers for the scanners
private static var scanner1Timer:Timer = new Timer(1000, 0);//scanner1 checks
one box every 1 sec
private static var scanner2Timer:Timer = new Timer(500/2, 0);//scanner2 checks
one box every 0.25 sec

//value of the midpoint of the address boxes
private static var midpointBoxes:Array = new Array();

//floor, ceiling and walls:
public static const floorYVal:Number = new Number(600);//floor y coordinate !!!
//Flash's coords are upside down...)
private static var screen:Screen = new Screen();//cover to make mouse actions
unavailable

//game state
private static var click:Boolean = false; //no click
private static var gameData:GameData = new GameData(screen);//,
```

```
  getMainInstance()); //points and current level of the game
  private static var gameMode:String = new String("");

  //Items panel
  private static var itemsPanel:ItemsPanel = new ItemsPanel(gameData);
  private static var itemsTab:ItemsTab = new ItemsTab();

  //temps:
  private static var currentPerson:Person = new Person(false);
  private static var freeSpotsCount:int = 0;//number of free spots
  //circles that leave line in shorter amount of time than normal
  private static var numLeaveLine:int = 0;

  //testing: to be removed
  public static var awesome:int = 0;//for testing purposes
  public static var temp = getTimer();

  //separation outlines
  private static var wallH:SepOutline = new SepOutline(400, 230, 0);
  private static var wallV:SepOutline = new SepOutline(579, 50, 90, (1+(1/5)));

  private static var wallArray:Array = new Array();//will contain the two
  sepOutlines

  private static var wallArea3:Array = new Array();//stores the different zones
  created from the walls

  public static var testLevel:int = 5;

//!Remove
  private static var nextLevel:NextLevel = new NextLevel();

  private static var messages:Messages = new Messages();//end of level messages

  /**
   * Sets up all of the things needed to start the game, such as reset exp,
   coins, player
   * lives, and sets the game mode.
   */
  public function setUpGame():void {
    //remove pesky yellow outline around buttons
    stage.addEventListener(Event.ACTIVATE, activated);
    //stage.addEventListener(Event.ENTER_FRAME, gameLoop);
    resetAll();//reset items

    addWalls();
//?
  pauseTimer = 0;//reset timer
    setUpPointsFormat();
    setUpLevelTextFormat();
    addChild(container);
    addChild(peopleContainer);
    addChild(boxTopsContainer);
    miscContainer.x = 0;
    miscContainer.y = 0;
```

```
    addChild(miscContainer);
//! remove
miscContainer.addChild(nextLevel);
nextLevel.buttonMode = true;
nextLevel.addEventListener(MouseEvent.MOUSE_DOWN, nextLevelListener);
//nextLevel.visible = false;
nextLevel.alpha = 0;

    addTable();//add desk to stage
    addForm();//add form
    addHourglass(); //add hourglass

    addAddresses();//set up boxes

    setUpPoints();//score keeping
    setUpLevelText();//keep track of the current level

    setUpScreen();
    setUpPanel();
    setUpTab();

    //make game responsive to key presses
    stage.addEventListener(KeyboardEvent.KEY_UP, pauseGame);
    stage.addEventListener(KeyboardEvent.KEY_DOWN, spendHourglassKeyPressHandler);

    levelCompleteScreen();
//setUpMessages();

    startGameLevel(100, 1);
  }

  /**
   * Starts the current game level and sets up the objects in it.
   */
  public function startGameLevel(pts:int, level:int):void {
    if (level < 8) {
    clearStage();//clear all circle objects from the stage

    //make game responsive to key presses
    stage.addEventListener(KeyboardEvent.KEY_UP, pauseGame);
    stage.addEventListener(KeyboardEvent.KEY_DOWN,
    spendHourglassKeyPressHandler);

    //set up the gameData info
    gameData.setPoints(pts);
    gameData.setGameMode("play");
    gameData.setLevel(level);
    levelText.text = ("Level: " + level);

    //reset people arrays
    people = new Array();
    area2 = new Array();
    area3 = new Array();

    currentPerson = new Person(false);
```

```
    freeSpotsCount = 0;
    numLeaveLine = 0;
    awesome = 0;
    temp = getTimer();
    itemsPanel.hideItemsPanel(true);
    pauseTimer = 0;//reset timer

    setUpText();
    gameData.setNumVisibleBoxes(showBoxes(gameData.getLevel()));
    initDevArray();


    //for authorized people:
    //figure out the midpoint of the boxes' left edge
    midpointBoxes = new Array();
    midpointBoxes = XYCoords();//400,100

    //draw line segment (edge of address boxes) in area 3
    drawLineSeg(XYLineSegCoords(), lineSeg);

    myTimer = new Timer(20, 10); //delays movement of people on the line
    //game loop runs when the timer has finished
    myTimer.addEventListener(TimerEvent.TIMER_COMPLETE, gameLoop);

    //makes the timer for each successive circle creation
    //ie: time to wait between circles.
    timePersonCreation = makeTimer();

    //starts timer and delays the checking of gameLoop
    myTimer.start();

    //things that change, based on level, but that are not in Main.as
    gameData.setSpeedOfLineBasedOnLevel();

    //at level 3, give the player a free, slow scanner
    if ((gameData.getItemCountInInventory(1) <= 0) && (gameData.getLevel() == 3))
    {
      gameData.incrementItemCountInInventory(1);
    }

    //at level 4, give the player a free wall
    if ((gameData.getItemCountInInventory(2) <= 0) && (gameData.getLevel() == 4))
    {
      gameData.incrementItemCountInInventory(2);
    }
    }else {
    if (gameCompleteSplash) {//if winner screen exists
      gameCompleteScreen();
    }
    }
  }

  /**
   * To remove the yellow border around buttons.
   * @param  evt
```

```
   */
  private function activated(evt):void {
    stage.focus = null;
  }

//!remove
  /**
   * Sets up the event listeners for a specific boxtop.
   * @param  ind: index of boxtop
   */
  private function addListenerToBoxTops(ind:int):void {
    //make top of box clickable
    tops[ind].buttonMode = true;

    tops[ind].addEventListener(MouseEvent.CLICK, deletePersonHandler);
    tops[ind].addEventListener(MouseEvent.CLICK, fadeOutBoxTopHandler);
  }

  /**
   * Checks if the current level needs to be updated/increased.
   */
  public function checkLevel():void {
    //if the points reach 0, then end game
    if (gameData.getPoints() <= 0) {
    gameData.setGameMode("game over");
    //go to the next level
    }else if (((gameData.getPoints() >= 150) && (gameData.getLevel() == 1))
    || ((gameData.getPoints() >= 180) && (gameData.getLevel() == 2))
    || ((gameData.getPoints() >= 200) && (gameData.getLevel() == 3))
    || ((gameData.getPoints() >= 220) && (gameData.getLevel() == 4))
    || ((gameData.getPoints() >= 240) && (gameData.getLevel() == 5))
    || ((gameData.getPoints() >= 260) && (gameData.getLevel() == 6))) {
    gameData.setGameMode("level complete");
    //advance the frame of the messages panel
    messages.advanceFrame(gameData.getLevel());
    //set up texts
    levelText.text = "Level: " + gameData.getLevel();
    pointsTextScreen.text =  "Points: " + gameData.getPoints() +
    "\nNext Level: " + (gameData.getLevel() + 1);


    //if player wins game
    }else if (gameData.getPoints() >= 280){
    gameData.setGameMode("winner");
    }
  }

  /**
   * Performs all game tasks and keeps track of what is currently happening
   * in the game.
   */
  public function gameLoop(event:TimerEvent):void {
  //public function gameLoop(event:Event):void {
//trace("mode: " + gameData.getGameMode());
```

```
    //reset form is out of screen
    if (form.x <= 0) {
    //form coords
    form.x = 200;
    form.y = 602;
    }

    //sets game mode... for actions panel in frame1
    if (gameMode != "") {
    gameData.setGameMode(gameMode);
    }
    addAdmin();
    spreadInfection();
    infectAllInSubnet();

    walls();
    checkHourglass();
    refreshPoints();
    startScannerTimer();
    runNAC();//places NAC into play
    checkLevel();

    // only perform tasks if in play mode
    if (gameData.getGameMode() == "play") {
    area3FreeSpots();
    //if there aren't any people on the line
    //or if there are, but there is still room,
    //add more people to the line
    if (roomInLine()
    && ((people.length + area2.length) <  freeSpotsCount)
    && (!timePersonCreation.running) && (people.length < 5)) {

//!take out and add admin at the beginning of the level
      addPerson(false);//person can't be admin, because the level is too low

      //if currentPerson is in area2
      if (area2.indexOf(currentPerson) != -1) {
      nextInLine();
      }
      timePersonCreation.reset();
      timePersonCreation.start();
    }
    //go through all people that are still on the line and find out which ones
    //need to be moved to area2, need to move or are frustrated
    for (var i:int = 0; i < people.length; i++) {
      if ((people[i] != null) && people[i]) {
      peopleCollision();//person bumped into previous person?
      //(makes movement look more static - as if the people are actually waiting
      //in line)

      if (people[i].lineMove) {//person moving down line
        lineMovement(gameData.getSpeedOfLine(),i); //pace of line
      }
      checkFrustration(people[i]);
```

```
  //if person can become frustrated:
  if (people[i].frustratable) {
    frustration(people[i]); //set up the frustration levels
  }

  //move the person from people to area2
  //person must either be authorized or must be unauthorized and annoyed
  if (((people[i].authorized) || (people[i].mood == 3)) &&
  (people[i].movingToBox == -1) && (area3FreeSpots())) {
    //take out the mouseover hand icon:
    people[i].buttonMode = false;

    //index of current person
    var p:Person;

    p = people.splice(i,1)[0];//move person out of the people array
    area2.push(p);//move person into area2 array

    //figure out the steps to reach area3 from area2
    stepSize(p, 5, midpointBoxes);
  }
  }
}
//loop through all of the people in area2 and change their opacity
//to 0, if isFading is true
for (var j:int = 0; j < area2.length; j++) {
  if ((area2[j] != null) && area2[j]) {
  //find box for person, if not already found
  emptyAddressLocations(area2[j]);
  fade(area2[j], 5);
  moveToBox(area2[j]);
  }
}


changeColorAll();//check if people in area3 need their color to be changed

//go through all people in area2 and move them toward the address boxes
area2Movement();//check to see if possible to move people toward address boxes

//checks area3 devices and takes care of setting their time limits
devicesArea3();

//change color for devices, if they turn red and make sure
//that they only stay on the stage for the amount of time allowed
evalAuthorizedAndUnauthorized();

//!!! CHECK LVL HERE AND THEN SET TIMER ACCORDINGLY
//reset timer to wait for the next gameLoop
myTimer.reset();
myTimer.start();

}else if (gameData.getGameMode() == "game over") {//stop game where it
is/freeze game
if (gameOverSplash) {
```

```
      gameOverScreen();
      pauseGameManually("pauseThroughElse");
      stage.removeEventListener(KeyboardEvent.KEY_UP, pauseGame);
      gameMode = "play";
    }
    }else if (gameData.getGameMode() == "level complete") {//take player to next
    level
    if (levelCompleteSplash) {
      levelCompleteContainer.visible = true;
      stage.removeEventListener(KeyboardEvent.KEY_UP, pauseGame);
      pauseGameManually("pauseThroughElse");
//      pauseGameManually("pause");//will pause w/out gray screen
    }
    }else if (gameData.getGameMode() == "winner"){
    if (gameCompleteSplash) {//if winner screen exists
      gameCompleteScreen();
//?

//      pauseGameManually("pause");//will pause w/out gray screen
    }
    }

  }

  /**
   * When the current level is complete, the game will move to
   * the next level.
   */
  public function levelCompleteScreen():void {
    miscContainer.addChild(levelCompleteContainer);

    //set up coords of splash
    levelCompleteSplash.x = 154;
    levelCompleteSplash.y = 128;
    //add splash bg to stage
    levelCompleteContainer.addChild(levelCompleteSplash);

    //add points:
    //level complete splash screen text box
    pointsTextScreen.autoSize = TextFieldAutoSize.LEFT;
    pointsTextScreen.defaultTextFormat = pointsTextScreenFormat;
    pointsTextScreen.embedFonts = true;
    pointsTextScreen.selectable = false;
    pointsTextScreen.width = 140;
    pointsTextScreen.x = 432;
    pointsTextScreen.y = 230;

    //set up text
    pointsTextScreen.text =  "Points: " + gameData.getPoints() +
    "\nNext Level: " + (gameData.getLevel() + 1);
    //add to stage
    levelCompleteContainer.addChild(pointsTextScreen);

    //add buttons to splash screen:
    //continue button:
```

```
    var continueButton:ContinueButton = new ContinueButton();
    continueButton.x = 414;
    continueButton.y = 298;

    //add to stage
    levelCompleteContainer.addChild(continueButton);

    //back to main menu button:
    var mainMenuButton2:MainMenuButton = new MainMenuButton();
    mainMenuButton2.x = 414;
    mainMenuButton2.y = 371;
    levelCompleteContainer.addChild(mainMenuButton2);

    //set up event listeners for the buttons:
    //continue button:
    continueButton.addEventListener(MouseEvent.CLICK, continueGame);

    //back to main menu button:
    mainMenuButton2.addEventListener(MouseEvent.CLICK, goToMain);

    //add the messages panel to show you what to expect for the next level
    miscContainer.addChild(messages);
    messages.returnButton().addEventListener(MouseEvent.CLICK, messagesContinueGame);

    levelCompleteContainer.visible = false;//make it invisible, temporarily
  }

  /**
   * Sets up/clears the stage in order for the player to continue
   * on to the next level.
   *
   * @param  e  mouse click
   */
  public function continueGame(e:MouseEvent):void {
    //update level
    gameData.setLevel(gameData.getLevel() + 1);
    levelText.text = "Level: " + gameData.getLevel();

    //make the messages panel visible (since done with this panel)
    messages.visible = true;
    //hide this panel
    levelCompleteContainer.visible = false;
  }

  /**
   * Continues the game, after the messages panel has appeared
   *  and the player has decided to continue the game.
   * @param  e:  Mouse Click
   */
  public function messagesContinueGame(e:MouseEvent):void {
    //reset game for next level
    startGameLevel(gameData.getPoints(), gameData.getLevel());

    //reset all visibilities to wait for next level upgrade
    messages.visible = false;
```

```
    //levelCompleteContainer.visible = true;
//*causing weird bug:
    pauseGameManually("playItems");
  }

  /**
   * Sets up the "you win" splash screen.
   */
  public function gameCompleteScreen():void {
    miscContainer.addChild(gameCompleteContainer);
    gameCompleteSplash.x = 160;
    gameCompleteSplash.y = 140;
    gameCompleteContainer.addChild(gameCompleteSplash);
    gameCompleteSplash.gotoAndPlay("winner");

    //add buttons to splash screen:
    //play again button:
    var playAgainButton2:PlayAgainButton = new PlayAgainButton();
    playAgainButton2.x = 414;
    playAgainButton2.y = 450;

    //add to stage
    gameCompleteContainer.addChild(playAgainButton2);

    //play again button:
    playAgainButton2.addEventListener(MouseEvent.CLICK, playAgain);

    pauseGameManually("pauseThroughElse");
  }

  /**
   * Clears the stage from circle objects.
   */
  public function clearStage():void{
    people = new Array();
    area2 = new Array();
    area3 = new Array();

    //remove the people that are still left over on the stage
    while (peopleContainer.numChildren > 0){
    peopleContainer.removeChildAt(0);
    }
  }

  //INIT. FUNCTIONS
  /**
   * Sets up the textboxes.
   */
  public function setUpText():void {
    //game paused text properties
    gamePauseText.x = 356.95;
    gamePauseText.y = 248;
    gamePauseText.visible = false;

    miscContainer.addChild(gamePauseText);
```

```
  }

  /**
   * Sets up the creation of the text that will show how many
   * points someone currently has.
   */
  public function setUpPoints():void {
    //points text box
    pointsText.autoSize = TextFieldAutoSize.LEFT;
    pointsText.defaultTextFormat = pointsTextFormat;
    pointsText.embedFonts = true;
    pointsText.selectable = false;
    pointsText.width = 140;
    pointsText.x = 5;
    pointsText.y = 5;

    //set up text
    pointsText.text =  "Points: " + gameData.getPoints();

    //add to stage
    container.addChild(pointsText);
  }

  /**
   * Sets up the formatting for all of the point
   * text boxes.
   */
  private function setUpPointsFormat():void {
    //font to embed
    var pointsFont:Courier = new Courier();

    //points text (topmost left) on game screen
    pointsTextFormat.font = pointsFont.fontName;

    //formatting:
    pointsTextFormat.color = 0x000000;
    pointsTextFormat.size = 30;
    //---------------------------------------
    //points text on game over screen
    pointsTextScreenFormat.font = pointsFont.fontName;

    //formatting:
    pointsTextScreenFormat.color = 0x000000;
    pointsTextScreenFormat.size = 25;
  }

//*
  /**
   * Sets up the current level counter.
   */
  public function setUpLevelText():void {
    //level text box
    levelText.autoSize = TextFieldAutoSize.LEFT;
    levelText.defaultTextFormat = pointsTextFormat;
    levelText.embedFonts = true;
```

```
    levelText.selectable = false;
    levelText.width = 140;
    levelText.x = 840;
    levelText.y = 5;

    //set up text
    levelText.text =  "Level: " + gameData.getLevel();

    //add to stage
    container.addChild(levelText);
  }

  /**
   * Sets up the formatting for the level text.
   */
  private function setUpLevelTextFormat():void {
    //font to embed
    var courierFont:Courier = new Courier();

    //points text (topmost left) on game screen
    levelTextFormat.font = courierFont.fontName;

    //formatting:
    levelTextFormat.color = 0xFFFFFF;
    levelTextFormat.size = 23;
    levelTextFormat.bold = true;
  }

  /**
   * Set up the items panel.
   */
  public function setUpPanel():void {
    itemsPanel.hideItemsPanel(true);
    miscContainer.addChild(itemsPanel);
    miscContainer.addChild(itemsPanel.getBgContainer());
  }

  /**
   * Sets up the tab of the items panel
   * and its event listener
   */
  public function setUpTab():void {
    itemsTab.x = 893.65;
    itemsTab.y = 110;
    itemsTab.useHandCursor = true;
    itemsTab.buttonMode = true;

    itemsTab.addEventListener(MouseEvent.CLICK, hideItemsPanel);
    itemsTab.addEventListener(MouseEvent.MOUSE_OVER, mouseOverTab);
    itemsTab.addEventListener(MouseEvent.MOUSE_OUT, mouseOutTab);

    miscContainer.addChild(itemsTab);
  }

  /**
```

```
 * Sets up the grey screen that will show up when game is paused
 * or when the items panel is called up.
 */
public function setUpScreen():void {
  screen.x = 0;
  screen.y = 0;
  screen.visible = false;

  miscContainer.addChild(screen);

}

/**
 * Creates the game over screen.
 */
private function gameOverScreen():void {
  //add the gameOverContainer to miscContainer
  miscContainer.addChild(gameOverContainer);

  //set up coords of splash
  gameOverSplash.x = 252;
  gameOverSplash.y = 133;
  //add splash bg to stage
  gameOverContainer.addChild(gameOverSplash);

  //add points:
  //game over splash screen text box
  pointsTextScreen.autoSize = TextFieldAutoSize.LEFT;
  pointsTextScreen.defaultTextFormat = pointsTextScreenFormat;
  pointsTextScreen.embedFonts = true;
  pointsTextScreen.selectable = false;
  pointsTextScreen.width = 140;
  pointsTextScreen.x = 432;
  pointsTextScreen.y = 220;

  //set up text
  pointsTextScreen.text =  "Points: " + gameData.getPoints() +
  "\nLevel: " + gameData.getLevel();

  //add to stage
  gameOverContainer.addChild(pointsTextScreen);

  //add buttons to splash screen:
  //play again button:
  var playAgainButton:PlayAgainButton = new PlayAgainButton();
  playAgainButton.x = 414;
  playAgainButton.y = 289;

  //add to stage
  gameOverContainer.addChild(playAgainButton);

  //back to main menu button:
  var mainMenuButton:MainMenuButton = new MainMenuButton();
  mainMenuButton.x = 414;
  mainMenuButton.y = 363;
```

```
    gameOverContainer.addChild(mainMenuButton);

    //set up event listeners for the buttons:
    //play again button:
    playAgainButton.addEventListener(MouseEvent.CLICK, playAgain);

    //back to main menu button:
    mainMenuButton.addEventListener(MouseEvent.CLICK, goToMain);
  }

  /**
   * Sets up/clears the stage in order for the player to play again.
   *
   * @param  e  mouse click
   */
  public function playAgain(e:MouseEvent):void {
//trace("clicked");
    //hide containers
    gameCompleteContainer.visible = false;
    gameOverContainer.visible = false;

    pauseGameManually("play");

    removeAllObjects();

    setUpGame();
    startGameLevel(100, 1);

  }

  /**
   * Takes the player back to the main game screen.
   *
   * @param  e  mouse click
   */
  public function goToMain(e:MouseEvent):void {
    removeAllObjects();//removes all objects from the stage and arrays
    gameData.setGameMode("stop");//reset game mode so that game doesn't play again
    this.gotoAndPlay(1);
  }

  /**
   * Adds the table object to the stage.
   */
  public function addTable():void{
    table = new Table();
    //table coords
    table.x = 0;
    table.y = 475;
    container.addChild(table);
  }

  /**
   * Adds the form object to the stage.
   */
```

```
public function addForm():void{
  form = new Form();
  //form coords
  form.x = 200;
  form.y = 602;
  container.addChild(form);
}

public function addHourglass():void {
  hourglass = new Hourglass();
  //hourglass coords
  hourglass.x = 888.75;
  hourglass.y = 482.25;
  //scaling
  hourglass.height = 102.30;
  hourglass.width = 54.05;
  container.addChild(hourglass);
  hourglass.alpha = 0;//make item invisible (since, at lvl 1, it's not available)

  hourglassMouseOver.x = 650;
  hourglassMouseOver.y = 484;
  hourglassMouseOver.visible = false;
  hourglassMouseOver.visible = false;
  container.addChild(hourglassMouseOver);

  //mouse over text for hourglass
  hourglass.addEventListener(MouseEvent.MOUSE_OVER, hourglassMouseOverListener);
  hourglass.addEventListener(MouseEvent.MOUSE_OUT, hourglassMouseOutListener);

  //add event listeners to houglass mouseover too,
  //otherwise, get weird effect if mouse is over mouseover
  hourglassMouseOver.addEventListener(MouseEvent.MOUSE_OVER,
  hourglassMouseOverListener);
  hourglassMouseOver.addEventListener(MouseEvent.MOUSE_OUT,
  hourglassMouseOutListener);

  //hourglass text:
  //format
  var hourglassFont:Courier = new Courier();
  hourglassInventoryTextFormat.font = hourglassFont.fontName;
  hourglassInventoryTextFormat.size = 20;

  hourglassInventoryText.x = 905;
  hourglassInventoryText.y = 559;
  hourglassInventoryText.width = 35;
  hourglassInventoryText.height = 25;
  hourglassInventoryText.background = true;
  hourglassInventoryText.backgroundColor = 0xFFFFFF;
  hourglassInventoryText.border = true;
  hourglassInventoryText.defaultTextFormat = hourglassInventoryTextFormat;
  hourglassInventoryText.text = "0";
  container.addChild(hourglassInventoryText);

  hourglassInventoryText.alpha = 0;//make item invisible (since, at lvl 1, it's
  not available)
```

```
    //add event listeners for the hourglass
    hourglass.addEventListener(MouseEvent.CLICK, spendHourglassHandler);
    hourglass.buttonMode = true;
    hourglassInventoryText.addEventListener(MouseEvent.CLICK, spendHourglassHandler);
    hourglassInventoryText.mouseEnabled = true;
  }

  /**
   * Listener to show mouseover text for hourglass.
   * @param  e: Mouse Over
   */
  public function hourglassMouseOverListener(e:MouseEvent):void {
    //only show text, if hourglass is visible
    if (hourglass.alpha == 1){
    hourglassMouseOver.visible = true;
    }
  }

  /**
   * Listener to hide mouseout text for hourglass.
   * @param  e: Mouse Out
   */
  public function hourglassMouseOutListener(e:MouseEvent):void {
    hourglassMouseOver.visible = false;
  }

  /**
   * Adds all of the address boxes for the current level, in such a way as to
   * maximize space, depending on the number of addresses that are available at
   that level.
   */
  public function addAddresses():void {
    //add the boxes to the addresses array
    //row 1
    addBox(400, 50);
    addBox(460, 50);
    addBox(520, 50);
    addBox(580, 50);
    addBox(640, 50);
    //row 2
    addBox(400, 110);
    addBox(460, 110);
    addBox(520, 110);
    addBox(580, 110);
    addBox(640, 110);
    //row 3
    addBox(400, 170);
    addBox(460, 170);
    addBox(520, 170);
    addBox(580, 170);
    addBox(640, 170);
    //row 4
    addBox(400, 230);
    addBox(460, 230);
    addBox(520, 230);
```

```
    addBox(580, 230);
    addBox(640, 230);
    //row 5
    addBox(400, 290);
    addBox(460, 290);
    addBox(520, 290);
    addBox(580, 290);
    addBox(640, 290);
    //row 6
    addBox(400, 350);
    addBox(460, 350);
    addBox(520, 350);
    addBox(580, 350);
    addBox(640, 350);

    //add box tops to the boxes
    //row 1
    addBoxTop(400, 50);
    addBoxTop(460, 50);
    addBoxTop(520, 50);
    addBoxTop(580, 50);
    addBoxTop(640, 50);
    //row 2
    addBoxTop(400, 110);
    addBoxTop(460, 110);
    addBoxTop(520, 110);
    addBoxTop(580, 110);
    addBoxTop(640, 110);
    //row 3
    addBoxTop(400, 170);
    addBoxTop(460, 170);
    addBoxTop(520, 170);
    addBoxTop(580, 170);
    addBoxTop(640, 170);
    //row 4
    addBoxTop(400, 230);
    addBoxTop(460, 230);
    addBoxTop(520, 230);
    addBoxTop(580, 230);
    addBoxTop(640, 230);
    //row 5
    addBoxTop(400, 290);
    addBoxTop(460, 290);
    addBoxTop(520, 290);
    addBoxTop(580, 290);
    addBoxTop(640, 290);
    //row 6
    addBoxTop(400, 350);
    addBoxTop(460, 350);
    addBoxTop(520, 350);
    addBoxTop(580, 350);
    addBoxTop(640, 350);
}

/**
```

```
 * Creates boxes, based on specification, and places them into
 * the addresses array, as well as adds them to the stage.
 *
 * @param  x:  x coordinate
 * @param  y:   y coordinate
 */
public function addBox(x:Number, y:Number):void {
  var box:Box = new Box();
  box.init(x, y);
  box.update(x, y);
  box.alpha = 0;//invisible
  container.addChild(box);
  addresses.push(box);
}

/**
 * Adds all of the box covers to the boxes.
 *
 * @param  x:  x coordinate
 * @param  y:   y coordinate
 */
public function addBoxTop(x:Number, y:Number):void{
  var top:BoxTop = new BoxTop();
  top.init(x, y);
  top.update(x, y);
  top.alpha = 0;
  boxTopsContainer.addChild(top);
  tops.push(top);
}

/**
 * Makes the boxes become visible, based on the current level.
 * (5 more boxes shown per level)
 *
 * @param  level:  current level of the game
 * @return  the number of currently visible boxes
 */
public function showBoxes(level:int):int {
  var num:int = 0;
  switch(level) {
  case 1:
    num = 5;
    break;
  case 2:
    num = 15;
    break;
  case 3:
    num = 25;
    break;
  case 4:
    num = 25;
    break;
  case 5:
    num = 30;
    break;
```

```
    case 6:
      num = 30;
      break;
    case 7:
      num = 30;
      break;
    }

    //show boxes needed
    for (var i:int = 0; i < num; i++) {
    addresses[i].alpha = 1;
//    addresses[i].visible = true;
    tops[i].alpha = 1;
    tops[i].visible = true;
    addListenerToBoxTops(i);
    tops[i].gotoAndStop(1);
    }
/*
    //hide boxes that are not needed
    for (var j:int = num; j < 30; j++) {
    addresses[i].visible = false;
    tops[i].visible = false;
    }
*/
    return num;
  }

  /**
   * Adds all of the people with devices that need to be registered onto the network.
   * Also assigns them all of their properties.
   */
  public function addPerson(b:Boolean):void {
    //figure out the range of the amount of time that it will take a circle to
    //turn from gray to blue-gray:
    var min:int;
    var max:int;

    //find out whether circle will have a shorter time to change to blue-gray
    //than the normal circles
    var rand:int = Math.floor(Math.random() * 30);
//*change commnets below:
    switch(gameData.getLevel()) {
    case 1://level 1: 5 circles per line movement
      //for level 1, rand must equal 0-5 for a circle to jump the line.
      if ((rand > 0) && (rand <= 3)) {
      min = -1000;
      max = 6000;
      }else {
      min = 17000;
      max = 17000;
      }
      break;

    case 2://15 circles per movement
      //for level 2, rand must equal 0-15 for a circle to jump the line.
```

```
  if ((rand > 0) && (rand <= 5)) {
  min = -2000;
  max = 4000;
  }else {
  min = 20000;//!!!
  max = 20000;//!!!
  }
  break;

case 3://25 circles per movement
  //for level 2, rand must equal 0-10 for a circle to jump the line.
  if ((rand > 0) && (rand <= 10)) {
  min = -2000;
  max = 4000;
  }else {
  min = 20000;//!!!
  max = 20000;//!!!
  }
  break;

case 4://25 circles per movement
  //for level 2, rand must equal 0-10 for a circle to jump the line.
  if ((rand > 0) && (rand <= 10)) {
  min = -2000;
  max = 4000;
  }else {
  min = 20000;//!!!
  max = 20000;//!!!
  }
  break;

case 5://30 circles per movement
  //for level 2, rand must equal 0-10 for a circle to jump the line.
  if ((rand > 0) && (rand <= 20)) {
  min = -2000;
  max = 4000;
  }else {
  min = 20000;//!!!
  max = 20000;//!!!
  }
  break;

case 6://30 circles per movement
  //for level 2, rand must equal 0-10 for a circle to jump the line.
  if ((rand > 0) && (rand <= 20)) {
  min = -2000;
  max = 4000;
  }else {
  min = 20000;//!!!
  max = 20000;//!!!
  }
  break;

case 7://30 circles per movement
  //for level 2, rand must equal 0-10 for a circle to jump the line.
```

```
      if ((rand > 0) && (rand <= 25)) {
      min = -2000;
      max = 4000;
      }else {
      min = 20000;//!!!
      max = 20000;//!!!
      }
      break;
    }

    var person:Person = new Person(b);
    if (!b) {//if not admin, make the person become purple
    person.timeToPurple(min,max);
    }

    //testing start
    person.awesome = awesome;//creation index
    awesome++;
    //testing end

    if (!b) {//if not admin, put the person on the line
    people.push(person);
    }

    peopleContainer.addChild(person);
  }

  /**
   * Initializes the devices array, which will contain the devices in their
   * proper address locations. From area3, you can, then, figure out which
   * device is in which address and which addresses locations are unoccupied.
   */
  private function initDevArray():void {
    for (var i:int = 0; i < addresses.length; i++) {
    if (addresses[i].alpha == 1) {//visible
      area3[i] = [null, false];//[obj, taken?]
    }
    }
  }


  //key presses (codes/functions)

  /**
   * Checks which key has been pressed.
   *
   * @param  event  key is pressed
   */
  public function pauseGame(event:KeyboardEvent):void {
//trace("key pressed: " + event.keyCode);
    //if not in play mode and the enter key is pressed, then
    //start game
    if ((gameData.getGameMode() != "play") && (event.keyCode == 13)){
    this.gotoAndStop("play");
```

```
  //if space bar is pressed, if game is being played, pause game
  }else if (event.keyCode == 32) {
  if (gameData.getGameMode() == "play") {
    gameData.setGameMode("pause");

    pauseGameItems("pause");

    myTimer.stop();
    showGreyScreen(true,true);//shows pause screen
    screen.addEventListener(MouseEvent.CLICK, screenClickHandler);
    pauseTimer = getTimer();//get the current time in order to record the
    amount of time that has lapsed

  //if game has been stopped, then continue game
  }else if (gameData.getGameMode() == "pause") {
    gameData.setGameMode("play");

    pauseGameItems("play");

    myTimer.start();
    screen.removeEventListener(MouseEvent.CLICK, screenClickHandler);
    showGreyScreen(false);//shows pause screen
    pauseTimer = getTimer() - pauseTimer; //stop the timer and record elapsed
    time
  }
  }
}

/**
 * Pause the game through the items panel.
 * @param  str:  what to do
 */
public function pauseGameManually(str:String):void {
  if (str == "pauseThroughItemsPanel") {
  gameData.setGameMode("pauseThroughItemsPanel");

  pauseGameItems("pauseThroughItemsPanel");

  myTimer.stop();
  showGreyScreen(true);//shows pause screen
  pauseTimer = getTimer();//get the current time in order to record the amount
  of time that has lapsed

  }else if (str == "play") {
  gameData.setGameMode("play");

  pauseGameItems("play");

  myTimer.start();
  showGreyScreen(false);//shows pause screen
  pauseTimer = getTimer() - pauseTimer; //stop the timer and record elapsed time
  }else if (str == "playItems"){
  gameData.resetScannerCurrentIndex();
  }else {
  gameData.setGameMode("pauseThroughElse");
```

```
    pauseGameItems("pauseThroughElse");

    myTimer.stop();
    showGreyScreen(false);//shows pause screen
    pauseTimer = getTimer();//get the current time in order to record the amount
    of time that has lapsed
    }
}

/**
 * Disable player from clicking on anything other than the Screen item.
 *
 * @param  e  Mouse click
 */
private function screenClickHandler(e:MouseEvent):void {
    e.target.useHandCursor = false;
    e.target.buttonMode = false;
}

//form
/**
 * Clicking on person leads to movement of form toward person and then back
 *
 * @param  event  MouseClick
 */
private function clickHandlerForm(event:MouseEvent):void {
    timeForm.addEventListener(TimerEvent.TIMER, timerHandlerForm);
    timeForm.start(); //starts timer
    //person is not frustratable anymore
    event.target.stopFrustration= true;
}

/**
 * Takes care of moving the form from the desk to the person waiting for it.
 *
 * @param  e  Timer
 */
private function timerHandlerForm(e:TimerEvent):void {
    if (currentPerson.mood < 3){
    currentPerson.moving = false; //waiting for form
    currentPerson.lineMove = false; //not moving while in line
    //if the timer has stopped running, reset & start it
    timeForm.reset();
    timeForm.start();
    moveForm(currentPerson.formMovement);
    if (!timeForm.running) {
      timeForm.reset();
      currentPerson.formMovement = "back";
      timeForm.start();
    }else {
      currentPerson.frustratable = false;
    }
    }else if (currentPerson.mood == 3){
    currentPerson.buttonMode = false;
```

```
  currentPerson.removeEventListener(MouseEvent.CLICK, clickHandlerForm);
  }
}

/**
 * Check all address boxes to see if contents are gray and, if so, figure out
 * whether they need to be changed to red or not.
 *
 * @param  e  Timer
 */
private function boxColorCheckHandler(e:TimerEvent):void {
  for (var i:int = 0; i < area3.length; i++) {
  if (area3[i] != null) {
    if (area3[i].mood == 0) {//if gray
    var num:Number = Math.floor(Math.random() * 10);
    //1/2 chance that the person will turn to red
    if (num <= 4) {
      changeColor(3, "both", area3, i); //change i's color to red
    }
    }
  }
  }
}

/**
 * Deletes the person that the player clicked on.
 *
 * @param  e  MouseClick
 */
public function deletePersonHandler(e:MouseEvent):void {
  //index of object clicked on
  var indx = tops.indexOf(e.currentTarget);

  //obj is found
  if (indx != -1) {
  if ((area3[indx][0] != null) && area3[indx][0]) {
    if (//((tops[indx].alpha >= 0) && (tops[indx].alpha <= 0.5))
    //||
    ((tops[indx].currentFrame >= 2) && (tops[indx].currentFrame <= 55))
    || (area3[indx][0].getAdmin())) {
    //if in debug/testing mode, give us all of the info on the person being
    removed
    allocatePoints(area3[indx][0], e);
    removeObject(peopleContainer, area3, indx);//delete obj

    //make top cover fully visible
    tops[indx].alpha = 1;//visible
    tops[indx].gotoAndPlay(76);//start visible tween
    }
  }
  }
}

/**
 * Takes care of spending one hourglass per mouse click on the hourglass item.
```

```
   * (Can't use more than one hourglass at a time -> makes game harder)
   * @param  e  MouseClick
   */
  public function spendHourglassHandler(e:MouseEvent):void {
    //if hourglass has been bought, is not running, and the game is in play mode
    if ((gameData.getItemCountInInventory(0) > 0) &&
    (!gameData.getHourglassTimer().running)
    && (gameData.getGameMode() == "play")) {
    gameData.decrementItemCountInInventory();
    gameData.slowDownLineSpeed();
    }
  }

  /**
   * Takes care of spending one hourglass when the h key is pressed.
   * (Can't use more than one hourglass at a time -> makes game harder)
   * @param  e  "h" key (code 72)
   */
  public function spendHourglassKeyPressHandler(e:KeyboardEvent):void {
    //if "h" key is pressed do the same as if there was a mouse click.
    if (e.keyCode == 72) {
    //if hourglass has been bought, is not running, and the game is in play mode
    if ((gameData.getItemCountInInventory(0) > 0) &&
    (!gameData.getHourglassTimer().running)
    && (gameData.getGameMode() == "play")) {
      gameData.decrementItemCountInInventory();
      gameData.slowDownLineSpeed();
    }
    }
  }

  /**
   * Allocates points based what object the user has decided to
   * boot from the address space.
   *
   * @param  obj  person to remove
   */
  public function allocatePoints(obj:Person, e:MouseEvent):void {
    //if the object is either gray or red
    if ((obj.mood == 0) || (obj.mood == 4)) {
    if (obj.getAdmin()) {
      removePoints(50);//if remove admin, then subtract 50 points
      var pb:PointBurst = new PointBurst(this,"-"+50,
      obj.x,25);
    }else {
      //difference between when the object was inserted into
      //its box and the current time
      var diff:Number = getTimer() - obj.boxTimer;

      //if the obj is gray (has not been infected)
      if ((diff <= 10000) && (obj.mood == 0)) {//give points to player
      if (e.type == "click"){
        addPoints(Math.floor((10000 - diff) / 1000) + 1);
trace("rem 7");
        var pb:PointBurst = new PointBurst(this,"+"+(Math.floor((10000 -
```

```
        diff)/1000)+1),
        obj.x, 25);
      }

      //if the obj is red, and its timer has not expired
      }else if ((diff <= 10000) && (obj.mood == 4)) {//remove points
      removePoints(Math.floor((diff - 10000) / 1000));
      //removePoints(Math.abs(Math.floor((10000 - diff) / 3000)) + 1);
trace("rem 8");
      var pb:PointBurst = new PointBurst(this,"+"+(Math.abs(Math.floor((10000 -
      diff)/3000))+1),
      obj.x, 25);
//***

      //if the object is red and the difference is between the period allowed,
      //but past the changing point of the object from gray to red
      }else if ((diff > 10000) && (diff <= obj.boxTimer)
      && ((diff - 10000) > 0) && (obj.mood == 4)) {

      var ind:int = -1;//index of object in area3
      //find the index of the object in area3
      for (var i:int = 0; i < area3.length; i++ ) {
        if (area3[i][0] != null) {
        if (area3[i][0] == obj) {
          ind = i;

        }
        }
      }

      var adminIndx:int = -1;//holds current admin's index
      adminIndx = returnAdminIndexForPerson(i);//get admin index

      //if admin (is red - defined above), then take off 10 extra points
      if ((adminIndx != -1)
      && (area3[returnAdminIndexForPerson(ind)][0].getAdmin())
      && (area3[returnAdminIndexForPerson(ind)][0].mood == 4)) {
        //take points away from the player
        removePoints(Math.floor((diff - 10000) / 1000) + 10);
trace("rem 5");
        var pb:PointBurst = new PointBurst(this,"-"+(Math.floor((diff - 10000) /
        1000) + 10),
        obj.x, 25);
//        trace("points removed2: " + (Math.floor((diff - 10000) / 1000) + 10));
      }else{
//***
        //take points away from the player
        removePoints(Math.floor((diff - 10000) / 1000));
trace("rem 6");
        var pb:PointBurst = new PointBurst(this,"-"+Math.floor((diff - 10000) /
        1000),
        obj.x, 25);
      }
      }
    }
```

```
    }else if (obj.mood == 1) {//object is green
trace("yes");
    //subtract 10 points for removing an authorized device
    removePoints(10);
trace("rem green2");

    var pb:PointBurst = new PointBurst(this, "-10", obj.x, 25);
trace("removed authorized device by clicking");
    }
  }

  /**
  * Finds all empty address boxes in area3, store their indices into an array,
  * pick one of them, randomly, based on array size, and register the object as
  * moving to it.
  *
  * @param   p:  person
  */
  private function emptyAddressLocations(p:Person):void {
    //person is currently not moving to any box and there are free spots in the
    boxes
    if (p.movingToBox == -1) {
    if (!p.getAdmin()) {//if person is not admin
      var tempArray:Array = new Array(); //temp array to store empty indices
      for (var i:int = 0; i < area3.length; i++) {
      //if empty space ([null,false] in our case), push indx into new array
      if (area3[i][1] == false) {
        tempArray.push(i);
      }
      }
      //number of free spots in area3
      var len:Number = tempArray.length;
      //if room for more objects, then add them
      if (len>0){
      var num:Number = -1;
      //use a random nums gen to get the val of indx
      num = Math.floor(Math.random() * len);
      //find out what value is at that index (we don't have a range in area3)
      //and note where the person will be moving to:
      p.movingToBox = tempArray[num];//index at area3
      area3[tempArray[num]][1] = true;//spot taken!
      }
    }
    }
  }


  /**
   * If there is another person in line, then that person becomes currentPerson.
   *
   * @return whether there is another person in line
   */
  private function nextInLine():Boolean {
    if ((people[0] != null) && people[0]) {
    currentPerson = people[0];
```

```
  return true;
  }else {
  return false;
  }
}

//main functions

/**
 * Changes mouse look to hand, allowing player to send out form to person
 * at the desk.
 */
private function formClick():void {
  if (currentPerson.mood == 3){//annoyed
  //remove "clickability"
  currentPerson.buttonMode = false;
  }else{
  //for mouse over purposes... makes it look nicer
  currentPerson.buttonMode = true;
  }
  //if current person is unauthorized and is not annoyed, then make him/her
  clickable
  if ((!currentPerson.authorized) && (currentPerson.mood < 3)) {
  //on mouseclick, add listeners to form
  currentPerson.addEventListener(MouseEvent.CLICK, clickHandlerForm);
  }
}

/**
 * Moves form to person.
 *
 * @param str: whether form should move backwards or forwards
 */
private function moveForm(str:String):void {
  if (str == "forward"){//move form to person at desk
  if (!collision(form,people[0])){ //most recent person at desk
    form.x -= 10;
    form.y -= 10;
  }else {
    timeForm.stop();//stops the timer
  }
  }
  else if (str == "back") {//move form back to desk
  if (!lineCollision("Y", floorYVal, form)) { //if no collision between
  //form and floor of game
    form.x += 10;
    form.y += 10;
  }else {//if collision with floor
    timeForm.stop();//stops the timer
    //person becomes authorized when it registers its device - device cannot be
    //compromised
    currentPerson.authorized = true;
    changeColor(1, "both"); //green

    currentPerson.frustratable = false;//person can be frustrated again
```

```
    //no more form movement:
    currentPerson.formMovement = "none";

    //remove the form listeners
    currentPerson.removeEventListener(MouseEvent.CLICK, clickHandlerForm);
    timeForm.removeEventListener(TimerEvent.TIMER, timerHandlerForm);
    addPoints(5);//give player 5 points for authorizing this device
    var pb:PointBurst = new
    PointBurst(this,"+5",currentPerson.x,currentPerson.y);
  }
  }
}

/**
 * Determines the movement of the people, while they are on the line to the
 front desk.
 *
 * @param  diff:  y coord increment
 * @param  i:  person[i] that is moving
 */
public function lineMovement(diff:int, i:int):void
{
  //person i is moving:
  people[i].moving = true;
  //if there is no collision between the person and the table, the person
  //has not been authorized yet and it is not frustrated
  if ((!collision(people[i], table)) && (!people[i].authorized) &&
  (people[i].mood != 3)) {
  //person moves down to table
  people[i].y += diff;

  //if the circle passes the table's edge, move it up
  if (table.y-people[i].y < 25){
    people[i].y = 450;
  }
  //if there is a collision with the table and the current person has not signed
  }else if ((collision(people[i], table)) && (!people[i].authorized)) {
  people[i].moving = false;//person has stopped
  people[i].lineMove = false; //not in line anymore

  //!!!check if this is only needed for form
  currentPerson = people[0];//set the global var pertaining to the current
  person
  //!!!!if person is not getting compromised, then give them a form
  formClick();//move form after user has accepted person

  //if the person is annoyed and has not been authorized yet
  }else if ((people[i].mood == 3) && (!people[i].authorized)) {
  people[i].moving = false;
  people[i].lineMove = false;
  }
}

/**
 * Move all of the people that are currently in area2 toward
```

```
 * the address boxes in area3.
 */
private function area2Movement():void {
  for (var i:int = 0; i < area2.length; i++)  {
  if ((area2[i] != null) && (area2[i])) {
    //if the person collides with the boxes'
    //edge
    if (collision(lineSeg, area2[i]) && (area2[i].collisions == 0)){
    //set up attributes
    area2[i].collisions++;
    area2[i].moving = false;
    area2[i].isFading = true;

    }else if ((area2[i].collisions < 1) && (area2[i].movingToBox != -1)){
    //person i is moving:
    area2[i].moving = true;

    //move circle:
    //check y value:
    if (area2[i].stepY > 0) {//circle comes from above
      //check if y value of circle is greater than the midpoint's y value
      if (area2[i].y > midpointBoxes[1]) {
      //move circle back to right place
      area2[i].y = midpointBoxes[1];
      area2[i].stepY = 0;//no need to move anymore

      //if they're equal, set the next step to 0
      }else if (area2[i].y == midpointBoxes[1]) {
      area2[i].stepY = 0;//no need to move anymore

      }else {
      area2[i].y += area2[i].stepY;//move circle
      }

    }else if (area2[i].stepY < 0) {//circle comes from below
      if (area2[i].y < midpointBoxes[1]) {//if above midpoint location
      //move circle back to right place
      area2[i].y = midpointBoxes[1];
      area2[i].stepY = 0;//no need to move anymore
      }else if (area2[i].y == midpointBoxes[1]) {
      area2[i].stepY = 0;//no need to move anymore
      }else {
      area2[i].y += area2[i].stepY;//move circle
      }
    }

    //check x value:
    //if circle has moved too far or if it is in the right place
    if (area2[i].x >= (midpointBoxes[0] - 25)) {
      //move circle back to right place
      area2[i].x = midpointBoxes[0] - 25;
      area2[i].stepX = 0;//no need to move anymore
    }else {//move circle
      area2[i].x += area2[i].stepX;//move circle
    }
```

```
    }
  }
  }
}

/**
 * Moves the object from area2 to area3, into a random address box
 * that is predetermined. The object is also moved into the area3
 * array and its alpha value is reset to 1.
 *
 * @param  obj:  the person
 */
private function moveToBox(obj:Person):void {
  if (!obj.getAdmin()) {//if obj is not admin
  //collision and obj is transparent -> move to box
  if (obj.alpha == 0) {
    var m:int;
    var indx:int;
    var p:Person;

    indx = area2.indexOf(obj);//find index
    m = obj.movingToBox;//address box index
    if (indx != -1) {//if found
    //take out p and replace with null obj
    p = area2.splice(indx, 1)[0];
    //move obj to designated address
    p.x = addresses[m].coordX + 30;
    p.y = addresses[m].coordY + 30;
    //move obj to area3
    area3.splice(m, 1, [p, true]);
    area3[m][0].alpha = 1;//visible!

    //start counting time from entry into box
    area3[m][0].boxTimer = getTimer();

    //change obj color to make its real color undetectable
    changeColor(5, "both", area3, m);

    //change obj's mood to gray, if it is purple, or blue-gray
    if ((area3[m][0].mood == 3) || (area3[m][0].mood == 2)) {
      changeColor(0, "mood", area3, m);//make mood become gray
    }
    }
  }
  }

}

//color changes
/**
 * Cycles through the levels of frustration of the people waiting in line,
 changing their
 * frames accordingly.
 *
 * @param   p:  person to start getting frustrated
```

```
   */
  private function frustration(p:Person):void {
    //time diff to compare to deltaMood
    var diffTime:int = getTimer()-p.timer;

    //if the change in time is greater than or equal to the amount of time required
    //to change the mood from gray to blue-gray, then find out in which
    //mood stage the person is in and change it to the next one. Change the color
    too.
    if (p.mood <= 2) { //current mood=grey or green
    //if person should have already changed color and its mood is gray
    if ((diffTime >= p.deltaMood) && (p.mood == 0)) {//if gray
      //switch the device's color to blue-gray and the mood to 2
      changeColor(2, "both", people, people.indexOf(p));

    //if the person should already have turned purple (3 sec after turning gray)
    //and its mood is blue-gray
    }else if ((diffTime >= (p.deltaMood+3000)) && (p.mood == 2)) {// if blue-gray
      //switch the device's color to purple and the mood to 3
      changeColor(3, "both", people, people.indexOf(p));
    }
    }
  }
//!insert color change of A here
  /**
   * Change the color of the person based on its mood.
   *
   * @param   mood:   person's mood
   * @param   change:  the type of change: mood, color or both
   * @param   otherArgs:   [0] - array name  or [0] - currentPerson
   *       [1] - array index
   */
  private function changeColor(mood:int, change:String, ... otherArgs):void {
    var len:Number = otherArgs.length;
    var newColor:ColorTransform = new ColorTransform();//color change
    switch(mood) {
    case 0: //gray
      newColor.color = 0x999999;
      if (len < 2) {
      if (((change == "color") || (change == "both")) &&
      (!currentPerson.getAdmin())) {
        currentPerson.color = "gray";
      }
      if (((change == "mood") || (change == "both")) &&
      (!currentPerson.getAdmin())) {
        currentPerson.mood = 0;
      }

      }else {
      if (otherArgs[0] == area3) {
        //if area3 is the array, then go about it differently
        if (((change == "color") || (change == "both")) &&
        (!area3[otherArgs[1]][0].getAdmin())) {
        area3[otherArgs[1]][0].color = "gray";
        }
```

```
      if (((change == "mood") || (change == "both")) &&
      (!area3[otherArgs[1]][0].getAdmin())) {
      area3[otherArgs[1]][0].mood = 0;
      }
    }else {
      if (((change == "color") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].color = "gray";
      }
      if (((change == "mood") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].mood = 0;
      }
    }
    }
    break;
  case 1: //green
    newColor.color = 0x99FF99;
    if (len < 2) {
    if (((change == "color") || (change == "both")) &&
    (!currentPerson.getAdmin())) {
      currentPerson.color = "green";
    }
    if (((change == "mood") || (change == "both")) &&
    (!currentPerson.getAdmin())) {
      currentPerson.mood = 1;
    }
    }else {
    if (otherArgs[0] == area3) {
      //if area3 is the array, then go about it differently
      if (((change == "color") || (change == "both")) &&
      (!area3[otherArgs[1]][0].getAdmin())) {
      area3[otherArgs[1]][0].color = "green";
      }
      if (((change == "mood") || (change == "both")) &&
      (!area3[otherArgs[1]][0].getAdmin())) {
      area3[otherArgs[1]][0].mood = 1;
      }
    }else {
      if (((change == "color") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].color = "green";
      }
      if (((change == "mood") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].mood = 1;
      }
    }
    }
    break;
  case 2: //blue-gray (bg)
    newColor.color = 0x9999CC;
    if (len < 2) {
    if (((change == "color") || (change == "both")) &&
    (!currentPerson.getAdmin())) {
```

```
      currentPerson.color = "blue-gray";
    }
    if (((change == "mood") || (change == "both")) &&
    (!currentPerson.getAdmin())) {
      currentPerson.mood = 2;
    }
    }else {
    if (otherArgs[0] == area3) {
      //if area3 is the array, then go about it differently
      if (((change == "color") || (change == "both")) &&
      (!area3[otherArgs[1]][0].getAdmin())) {
      area3[otherArgs[1]][0].color = "blue-gray";
      }
      if (((change == "mood") || (change == "both")) &&
      (!area3[otherArgs[1]][0].getAdmin())) {
      area3[otherArgs[1]][0].mood = 2;
      }
    }else {
      if (((change == "color") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].color = "blue-gray";
      }
      if (((change == "mood") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].mood = 2;
      }
    }
    }
    break;
  case 3: //purple
    newColor.color = 0x993399;
    if (len < 2) {
    if (((change == "color") || (change == "both")) &&
    (!currentPerson.getAdmin())) {
      currentPerson.color = "purple";
    }
    if (((change == "mood") || (change == "both")) &&
    (!currentPerson.getAdmin())) {
      currentPerson.mood = 3;
    }
    var pb:PointBurst = new PointBurst(this,"-5",currentPerson.x,
    currentPerson.y);
    }else {
    if (otherArgs[0] == area3) {
      //if area3 is the array, then go about it differently
      if (((change == "color") || (change == "both")) &&
      (!area3[otherArgs[1]][0].getAdmin())) {
      area3[otherArgs[1]][0].color = "purple";
      }
      if (((change == "mood") || (change == "both")) &&
      (!area3[otherArgs[1]][0].getAdmin())) {
      area3[otherArgs[1]][0].mood = 3;
      }
      var pb:PointBurst = new PointBurst(this,"-5",area3[otherArgs[1]][0].x,
      area3[otherArgs[1]][0].y);
```

```
    }else {
      if (((change == "color") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].color = "purple";
      }
      if (((change == "mood") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].mood = 3;
      }
      var pb:PointBurst = new PointBurst(this,"-5",otherArgs[0]
      [otherArgs[1]].x,otherArgs[0][otherArgs[1]].y);
    }
    }

    //person becoming purple leads to less points
    //to keep player interested in registering people
    removePoints(5);
    break;
  case 4: //red
    newColor.color = 0xCC0033;
    if (len < 2) {
    if (((change == "color") || (change == "both")) &&
    (!currentPerson.getAdmin())) {
      currentPerson.color = "red";
    }
    if (((change == "mood") || (change == "both")) &&
    (!currentPerson.getAdmin())) {
      currentPerson.mood = 4;
    }
    }else {
    if (otherArgs[0] == area3) {
      //if area3 is the array, then go about it differently
      if (((change == "color") || (change == "both")) &&
      (!area3[otherArgs[1]][0].getAdmin())) {
      area3[otherArgs[1]][0].color = "red";
      }
      if (((change == "mood") || (change == "both")) &&
      (!area3[otherArgs[1]][0].getAdmin())) {
      area3[otherArgs[1]][0].mood = 4;
      }
    }else {
      if (((change == "color") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].color = "red";
      }
      if (((change == "mood") || (change == "both")) &&
      (!otherArgs[0][otherArgs[1]].getAdmin())) {
      otherArgs[0][otherArgs[1]].mood = 4;
      }
    }
    }

    break;
  }
  if ((change == "color") || (change == "both")) {
```

```
  if (len < 2) {
    if (!currentPerson.getAdmin()) {//if person is not admin
    currentPerson.transform.colorTransform = newColor;
    }
  }else{
    if (otherArgs[0] == area3) {
    //if area3 is the array, then go about it differently
    if (!area3[otherArgs[1]][0].getAdmin()) {
      area3[otherArgs[1]][0].transform.colorTransform = newColor;
    }
    }else {
    if (!otherArgs[0][otherArgs[1]].getAdmin()) {
      otherArgs[0][otherArgs[1]].transform.colorTransform = newColor;
    }
    }
  }
  }
}

//collisions:

/**
 * Checks whether an object collides with a given line. (object's x and/or
 * y values are taken into account, depending on the situation.)
 * Line's X or Y value is checked against X and Y values of the object.
 *
 * @param coord: X or Y coordinate of line
 * @param value: the value of the coordinate
 * @param obj: the object to be checked against the line
 */

public function lineCollision(coord:String, value:Number, obj:Form):Boolean {
  if (obj.y >= value) {
    return true;
  }else{
    return false;
  }
}

/**
 * Collision detection code (such as if person hits the wall of the addresses
 * or the desk).
 *
 * @param one: first movie clip to check collision with second movie clip
 * @param two: second movie clip to check collision with first movie clip
 */

public function collision(one:MovieClip, two:MovieClip):Boolean {
  if (one.hitTestObject(two)) {
  return true;
  }else{
  return false;
  }
}
```

```
/**
 * Check whether each consecutive two people in the line have collided with
 * each other or with the person next to him/her.
 * If so, stop the latter's movement.
 */
public function peopleCollision():void {
  //go through everyone in the line and check if any collisions
  for (var i:int = 1; i < people.length; i++){
  //vertical difference between two people
  var diff:Number = people[i - 1].y - people[i].y - 50;

  if (diff == 0) {
    //collision
    people[i].lineMove = false;//stop moving
    people[i].moving = false;//stop moving
  }else if (diff < 0) {
    //beyond collided, move to colliding position
    people[i].y = people[i].y - Math.abs(diff);
    people[i].lineMove = false;//stop moving
    people[i].moving = false;//stop moving
  }else{
    //no collision
    if ((!people[i].moving) && (!people[i].lineMove) &&
        (!collision(people[i], table))) {
    people[i].lineMove = true;//move
    people[i].moving = true;//move
    }
  }
  }
  //if 1st person is not moving, but there's no collision with table,
  //1st person should keep moving
  if ((!people[0].moving) && (!people[0].lineMove) && (!collision(people[0],
  table))) {
  people[0].lineMove = true;//move
  people[0].moving = true;//move
  }
}

/**
 * Intro screen code.
 */

/**
 * Tutorial level code.
 */

/**
 * Code to clean up what's left of the game after the game is over.
 */
private function cleanUp():void {

}

/**
 * Checks whether there is enough room in the line for another
```

```
 * person.
 *
 * @return  Boolean:    true = enough room
 *       false = not enough room
 */
private function roomInLine():Boolean {
  //if last person in line is too close to the ceiling
  if (people.length > 0) {//people are in line
  //last person's y coord
  if (people[people.length - 1].y < 100) {
    return false;
  }else {
    return true;
  }
  }else {//no one in line, there's room
  return true;
  }
}

/**
 * Returns whether there are any free spots in the address boxes.
 *
 * @return  true:  if free spot(s)
 */
private function area3FreeSpots():Boolean {
  freeSpotsCount = 0;
  for (var i:int = 0; i < area3.length; i++) {
  //if no registered object moving there
  if (area3[i][1] == false) {
    freeSpotsCount++;
  }
  }
  if (freeSpotsCount > 0){
  return true;//free spots
  }else{
  return false;//no free spots
  }
}

/**
 * Make all area3 objects clickable.
 *
 * @param  i  location of person
 */
public function clickableInBox(i:int):void {
  //if person cannot be clicked
  if ((area3[i][0] != null) && area3[i][0]) {
  if (!area3[i][0].buttonMode){
    //make person clickable
    area3[i][0].buttonMode = true;
  }
  }
}
```

```
  /**
   * Makes the top of the box that the player has clicked become
   * invisible
   *
   * @param  e  MouseClick Event
   */
  public function fadeOutBoxTopHandler(e:MouseEvent):void {
    var index:int = tops.indexOf(e.currentTarget);
    //if there is a person in box, change the person's color
    //to its "natural color"/mood, instead of the box's
    if ((area3[index][0] != null) && area3[index][0]) {
    changeColor(area3[index][0].mood, "both", area3, index);
    e.currentTarget.gotoAndPlay(2);

    //if no person in box, then open and close tween
    }else {
    e.currentTarget.gotoAndPlay(101);
    }
  }

  /**
   * Calculates how much time all devices can spend
   * in their respective boxes.
   *
   * @param  i  location of person
   */
  public function setDeviceTime(i:int):void {
    var time:Number = -1;
//!changed!
    var additionalTime:int = 1000;//time to add to timer that measures how long
    the person is staying for

    //if no time set yet, then set the time
    if (area3[i][0].periodAllowedInBox == -1) {
    if (area3[i][0].mood == 0) {//gray
      time = 0;
      var temp:Number = Math.floor(Math.random() * 2);//0 or 1 --> for color
      change
      if (temp == 0) {//red
      area3[i][0].unauthorizedParam[0] = "red";
      temp = Math.floor(Math.random() * 11);//for stay or leave
      if (temp < 5) {//leave
        area3[i][0].unauthorizedParam[1] = "leave";
      }else {//stay
        area3[i][0].unauthorizedParam[1] = "stay";
        time += temp * additionalTime;//add to time staying
      }
      }else if (temp == 1) {//stay gray
      area3[i][0].unauthorizedParam[0] = "not red";
      area3[i][0].unauthorizedParam[1] = "stay";
      //stay for this amount of time, then leave
      temp = Math.floor(Math.random() * 11) + 5;
      time += temp * additionalTime;
      }
    }else if (area3[i][0].mood == 1) {//green
```

```
      time = 20000; //20 sec, initially
      area3[i][0].unauthorizedParam[0] = "not red";
      area3[i][0].unauthorizedParam[1] = "stay";
    }
    area3[i][0].periodAllowedInBox = time;
    }
  }

  /**
   * Goes through the unauthorized devices in area3 and makes all
   * devices clickable, sets up how much time they can spend in
   * their box, and checks if they have overstayed their time
   * limit.
   */
  public function devicesArea3():void {
    for (var i:int; i < area3.length; i++) {//check all devices
    if ((area3[i][0] != null) && (area3[i][0]) && (!area3[i][0].getAdmin()))
    {//exists
      //if the time that the device is allowed to stay in the box
      //for has not been set yet, then set it
      if (area3[i][0].periodAllowedInBox == -1){
      //find out how long device will stay in box
      setDeviceTime(i);
      }
    }
    clickableInBox(i);//allow box tops to be moused over
    }
  }

  /**
   * Change the color of all of the area3 unauthorized devices to red,
   * if it was so decided and remove them from the stage, when the time
   * comes. Allow authorized devices to stay on the stage for their specified
   * amount of time allowed.
   */
  public function evalAuthorizedAndUnauthorized():void {
    var time:Number = 10000;//time to wait before becoming red
    //loop through the area3 people
    for (var i:int = 0; i < area3.length; i++) {
    if ((area3[i][0] != null) && area3[i][0]) {//contains person
      if (!area3[i][0].getAdmin()) { //if person is not an admin
      //find the elapsed time between the person going into the
      //address box and the current time
      var diff:Number = getTimer() - area3[i][0].boxTimer;
      if (area3[i][0].mood == 1) {//green - authorized device
        //object has overstayed its time limit
        if (diff >= area3[i][0].periodAllowedInBox) {
//*
        //give points to the player
        addPoints(10);
trace("rem green on its own");
        var pb:PointBurst = new PointBurst(this, "+10", area3[i][0].x, 25);
        removeObject(peopleContainer, area3, i);
        tops[i].gotoAndPlay(76);//make top cover fully visible
//trace("removed authorized device");
```

```
        }
      }else { //unauthorized device
        //wait 10 sec, then check if obj is turning red
        if (diff >= time) {//used to be 5000
        //if turning red and is already red, just check if has
        //overstayed allowed time frame
        if ((area3[i][0].unauthorizedParam[0] == "red") &&
        (area3[i][0].mood == 4)) {
          if ((diff - time) >= area3[i][0].periodAllowedInBox) {
          if ((diff > 10000) && (diff <= area3[i][0].boxTimer)
          && ((diff - 10000) > 0) && (area3[i][0].mood == 4)) {
            var adminIndx:int = -1;//holds current admin's index
            adminIndx = returnAdminIndexForPerson(i);//get admin index

            //if admin (is red - defined above), then take off 10 extra points
            if ((adminIndx != -1)
            && (area3[adminIndx][0].getAdmin())
            && (area3[adminIndx][0].mood == 4)) {
            //take points away from the player
            removePoints(Math.floor((diff - 10000) / 1000) + 10);
//trace("rem 1");
            var pb:PointBurst = new PointBurst(this,"-"+
            (Math.floor((diff - 10000) / 1000) + 10), area3[i][0].x, 25);
//            trace("points removed: " + (Math.floor((diff - 10000) / 1000) +
10));
            }else{
            //take points away from the player
            removePoints(Math.floor((diff - 10000) / 1000));
//trace("rem 2");
            var pb:PointBurst = new PointBurst(this,"-"+
            Math.floor((diff - 10000) / 1000), area3[i][0].x, 25);
            }
          }

          removeObject(peopleContainer, area3, i);

          tops[i].gotoAndPlay(76);//make top cover fully visible
          }
        //if supposed to turn red, but not red yet, change mood to red
        }else if ((area3[i][0].unauthorizedParam[0] == "red") &&
        (area3[i][0].mood != 4)) {
          changeColor(4, "mood", area3, i);//to test, use "both"
        //if not supposed to change to red:
        }else if (area3[i][0].unauthorizedParam[0] == "not red") {


          //!testing
          //  trace("index of admin: "+returnAdminIndexForPerson(i)[0]);



          //has it overstayed time limit?
          if ((diff - time) >= area3[i][0].periodAllowedInBox) {

          //if admin is red, then take off 10 extra points
```

```
          //subtract points, based on how long object has been there
          removePoints(Math.floor((diff - 10000) / 1000));
//trace("removed: " + Math.floor((diff - 10000) / 1000));
//trace("rem 3");
          var pb:PointBurst = new PointBurst(this,"-"+
          Math.floor((diff - 10000) / 1000),area3[i][0].x,25);

          removeObject(peopleContainer, area3, i);

          tops[i].gotoAndPlay(76);//make top cover fully visible
          }
        }
        }
      }
      }
    }
    }
  }

  /**
   * Removes the object from the stage as well as from area3 array.
   *
   * @param  arr:  array where the object is located
   * @param  i:  index of object in the array
   * @param   cont:  container where obj is stored
   */
  private function removeObject(cont:Sprite, ... args):void {
    if (args.length == 2) {
    if (args[0] == area3){
      cont.removeChild(area3[args[1]][0]);//remove obj from container
      //remove obj from area3 array
      area3.splice(args[1], 1, [null, false]);
    }else{
      cont.removeChild(args[0][args[1]]);//remove obj from container
      //remove obj from arr array
      args[0].splice(args[1], 1);
    }
    }else{
    cont.removeChild(args[0][args[1]]);//remove obj from container
    }
  }

  /**
   * Deletes all objects from the stage, containers and arrays.
   */
//!remove this function
  public function removeAllObjects():void {
    clearStage();//clear all circle objects from the stage

    //remove objects from the stage:
    //remove levelCompleteContainer (contains level complete screen)
    while (levelCompleteContainer.numChildren > 0){
    levelCompleteContainer.removeChildAt( 0 );
    }
```

```
  //remove gameCompleteContainer (contains game complete screen)
  while (gameCompleteContainer.numChildren > 0){
  gameCompleteContainer.removeChildAt( 0 );
  }

  //remove items from gameOverContainer
  while (gameOverContainer.numChildren > 0){
  gameOverContainer.removeChildAt( 0 );
  }

  //remove miscContainer (contains game over splash screen)
  while (miscContainer.numChildren > 0){
  miscContainer.removeChildAt( 0 );
  }

  //clear all other objects from the stage
  while (container.numChildren > 0){
  container.removeChildAt( 0 );
  }

  //remove boxTopsContainer (contains box tops)
  while (boxTopsContainer.numChildren > 0){
  boxTopsContainer.removeChildAt( 0 );
  }


  //clear all objects from their respective arrays
  //(circles have already been taken care of)

  //clear all items from the addresses array (boxes)
  if (addresses) {
  while (addresses.length > 0) {
    addresses.pop();
  }
  }

  //clear all items from the tops array
  if (tops) {
  while (tops.length > 0) {
    tops.pop();
  }
  }

}

/**
 * Checks if the person should be "frustratable" or not and sets
 * his/her parameter accordingly.
 *
 * @param  p:  Person
 */
public function checkFrustration(p:Person):void {
  //not moving and has not signed form
  var j:int=people.indexOf(p);
```

```
    //if stopFrustration is true, can't become frustrated
    if (p.stopFrustration) {
    p.frustratable = false;
    }else {//can become frustrated while moving
    if (!p.authorized) {////(!p.lineMove) && (!p.moving) &&
      p.frustratable = true;//can be frustrated
    }else {
      p.frustratable = false;//cannot be frustrated
    }
    }
  }
//!needs to be changed
  /**
   * Find out the X and Y coordinates of the boxes' line segment (person
   * will collide with them in order to take up an address location).
   *
   * @return array: with the following structure:
   *       [X_coord, Y_0_coord, Y_1_coord]
   */
  private function XYLineSegCoords():Array {
    var X:int;
    var Y_0:int;
    var Y_1:int;
    var xyarray:Array = new Array();

    X = addresses[0].coordX;
    Y_0 = addresses[0].coordY;
    Y_1 = addresses[0].coordY + (6 * 60);
    xyarray.push(X);
    xyarray.push(Y_0);
    xyarray.push(Y_1);
/*
    switch(gameData.getLevel()) {
    case 1: //level 1
      X = addresses[0].coordX;
      Y_0 = addresses[0].coordY;
      Y_1 = addresses[0].coordY + 60;
      xyarray.push(X);
      xyarray.push(Y_0);
      xyarray.push(Y_1);
      break;

    case 2: //level 2
      X = addresses[0].coordX;
      Y_0 = addresses[0].coordY;
      Y_1 = addresses[0].coordY + (5 * 60);
      xyarray.push(X);
      xyarray.push(Y_0);
      xyarray.push(Y_1);
      break;

    default: //levels 3 through 5
      X = addresses[0].coordX;
      Y_0 = addresses[0].coordY;
```

```
      Y_1 = addresses[0].coordY + (3 * 60);
      xyarray.push(X);
      xyarray.push(Y_0);
      xyarray.push(Y_1);
      break;
    }
*/
    //return the array of coords
    return xyarray;
  }

  /**
   * Find out the X and Y coordinate of the middle of the boxes' edge
   * (useful for finding how person will move to there).
   *
   * @return array: with the following structure:
   *       [X_coord, Y_coord]
   */
  private function XYCoords():Array {
    /*
    var rows:int;//number of rows
    var mid:int;//index of middle box
    var index:int;//index of the going to box

    rows = numVisibleBoxes / 5;
    mid = Math.ceil(rows / 2);
    index = (mid - 1) * rows + 1;
    */


    var X:Number;
    var Y:Number;
    var xyarray:Array = new Array();
    //xyarray.push(addresses[index].coordX);
    //xyarray.push(addresses[index].coordY + 30);//+half of box height


    switch(gameData.getLevel()) {
    case 1: //level 1
      X = addresses[0].coordX;
      Y = addresses[0].coordY + 30;//0.5 of box edge
      xyarray.push(X);
      xyarray.push(Y);
      break;

    case 2: //level 2
      X = addresses[0].coordX;
      Y = addresses[0].coordY + 90;
      xyarray.push(X);
      xyarray.push(Y);
      break;

    case 3: //level 3
      X = addresses[0].coordX;
      Y = addresses[0].coordY + 150;
```

```
      xyarray.push(X);
      xyarray.push(Y);
      break;

    case 4: //level 4
      X = addresses[0].coordX;
      Y = addresses[0].coordY + 150;
      xyarray.push(X);
      xyarray.push(Y);
      break;

    default: //levels 5-7
      X = addresses[0].coordX;
      Y = addresses[0].coordY + 210;
      xyarray.push(X);
      xyarray.push(Y);
      break;
    }
    //return the array of coords
    return xyarray;
  }
//*change this to make is slower/faster per level
  /**
   * Calculate the x and y step size for a specific person,
   * using the midpoint of the address box space as well as
   * the location of the person.
   *
   * @param  p:    person to be moved
   * @param  deltaT:  time it takes to move from
   *       area2 to area3
   * @param  coords:   X and Y values to get to
   *       (midpoint of address space)
   */
  private function stepSize(p:Person, deltaT:Number, coords:Array) {
    //steps have never been calculated
    if ((p.stepX == -1) && (p.stepY == -1)) {
    //calculate the step size for a circle
    if (p.y == coords[1]) {//circle is across box
      p.stepX = (coords[0] - p.x - 25) / deltaT;//-60
      p.stepY = 0;
    }else {//circle is above or under box
      p.stepX = (coords[0] - p.x - 25) / deltaT;
      p.stepY = (coords[1] - p.y) / deltaT;
    }

    }
  }

  /**
   * Fades the person until alpha=0.
   *
   * @param  obj:  person to be faded
   * @param  time:  time it should take to fade
   */
  private function fade(obj:Person, time:Number):void {
```

```
  if (obj.isFading) {//fade-mode
  obj.alpha -= 1 / time;//fade person by some degree

  //reset alpha to 0, if alpha goes below 0
  if (obj.alpha < 0) {
    obj.alpha = 0;
    obj.isFading = false;//obj has finished fading
  }
  }
}

/**
 * Draws a line segments on the left edge of the address boxes.
 *
 * @param  coords:   array of coordinates to draw the line segment
 */
private function drawLineSeg(coords:Array, name:MovieClip):void {
  name.graphics.lineStyle(1, 0xCC33CC, 0);
  name.graphics.moveTo(coords[0], coords[1]);
  name.graphics.lineTo(coords[0], coords[2]);
  container.addChild(name);
}

/**
 * Creates a timer based on the current level. The timer
 * serves to speed up or slow down the creation of circle
 * objects that appear in the line.
 *
 * @return  Timer
 */
private function makeTimer():Timer {
  var time:Timer;
  switch(gameData.getLevel()) {
  case 1:
    time = new Timer(10000,1);//(30000,1);
    break;

  case 2:
    time = new Timer(3000,1);
    break;

  case 3:
    time = new Timer(200,1);
    break;

  case 4:
    time = new Timer(200,1);
    break;

  case 5:
    time = new Timer(100,1);
    break;

  case 6:
    time = new Timer(100,1);
```

```
      break;

    case 7:
      time = new Timer(100,1);
      break;


    }
    return time;
  }

  /**
   * Adds points to the total amount of points and
   * updates the score text.
   *
   * @param  points  the number of points to be added
   */
  public function addPoints(points:Number):void {
    gameData.setPoints(gameData.getPoints() + points);//adds points to the score
    pointsText.text = "Points: " + gameData.getPoints();//update score text
    //setting format
    pointsText.setTextFormat(pointsTextFormat);
  }

  /**
   * Removes points from the total amount of points and
   * updates the score text.
   *
   * @param  points  the number of points to be removed
   */
  public function removePoints(points:Number):void {
    gameData.setPoints(gameData.getPoints() - points);//remove points from the
    score
    if (gameData.getPoints() < 0){
    gameData.setPoints(0);//reset points to 0
    }
    pointsText.text = "Points: " + gameData.getPoints();//update score text
    //setting format
    pointsText.setTextFormat(pointsTextFormat);
  }

  /**
   * Displays the current amount of points or refreshes the textbox that
   * displays these points.
   */
  public function refreshPoints():void {
    pointsText.text = "Points: " + gameData.getPoints();//update score text
    //setting format
    pointsText.setTextFormat(pointsTextFormat);
  }

  //functions pertaining to the items panel and items

  /**
   * Makes the panel visible or invisible, on mouseclick.
```

```
 *
 * @param  e:  mouse click
 */
public function hideItemsPanel(e:MouseEvent):void {
  //if panel is already hidden, then clicking on button will show panel
  if (itemsPanel.getHideState()) {
  itemsPanel.hideItemsPanel(false);
  pauseGameManually("pauseThroughItemsPanel");
  }else{
  itemsPanel.hideItemsPanel(true);
  pauseGameManually("play");
  }
}

/**
 * Changes the current frame of the ItemsPanel tab,
 * depending on the state of the panel (hidden or not hidden).
 * @param  e: mouse over
 */
public function mouseOverTab(e:MouseEvent):void {
  //if the panel is hidden, on mouse over go to frame 2
  if (itemsPanel.getHideState()) {
  itemsTab.gotoAndPlay(2);



  //if the panel is not hidden, on mouse over go to frame 4
  }else {
  itemsTab.gotoAndPlay(4);
  }
}

/**
 * Changes the current frame of the ItemsPanel tab,
 * depending on the state of the panel (hidden or not hidden).
 * @param  e: mouse out
 */
public function mouseOutTab(e:MouseEvent):void {
  //if the panel is hidden, on mouse out go to frame 1
  if (itemsPanel.getHideState()) {
  itemsTab.gotoAndPlay(1);

  //if the panel is not hidden, on mouse out go to frame 3
  }else {
  itemsTab.gotoAndPlay(3);
  }
}

/**
 * If the player tries to buy an item, it finds out if he/she can,
 * and, if so, it adds that item to the items array.
 */

 /*public function buyItem():void {
  //itemsPanel.buyItem();
```

```
  trace("hello");
}*/

/**
 * Shows the grey screen, along with or without the pause text.
 * @param  showScreen: whether or not to show the grey screen
 * @param  ... bools: [0] -> text boolean
 */
public function showGreyScreen(showScreen:Boolean, ... bools):void {
  //if showScreen is true, then show the grey screen
  screen.visible = showScreen;
  //show text only if it requested (array of 1 Boolean)
  gamePauseText.visible = bools[0];
}

/**
 * Check if the hourglass could be potentially used,
 * and if so, it makes the hourglass icon appear at
 * the bottom of the screen.
 */
public function checkHourglass():void {
  if (gameData.getLevel() > 1) {
  //make hourglass visible
  hourglass.alpha = 1;
  hourglassInventoryText.alpha = 1;

  //add event listeners for the hourglass and make it clickable
  hourglass.addEventListener(MouseEvent.CLICK, spendHourglassHandler);
  hourglass.buttonMode = true;
  hourglassInventoryText.addEventListener(MouseEvent.CLICK,
  spendHourglassHandler);
  hourglassInventoryText.mouseEnabled = true;

  hourglassInventoryText.text =
  (gameData.getItemCountInInventory(0)).toString();//refresh the amount of
  hourglass items in inv
  }else {
  //make hourglass invisible and unclickable
  hourglass.alpha = 0;
  hourglassInventoryText.alpha = 0;

  //add event listeners for the hourglass and make it clickable
  hourglass.removeEventListener(MouseEvent.CLICK, spendHourglassHandler);
  hourglass.buttonMode = false;
  hourglassInventoryText.removeEventListener(MouseEvent.CLICK,
  spendHourglassHandler);
  hourglassInventoryText.mouseEnabled = false;
  }


}

/**
 * Start one of the two scanners' timer.
 * @param  scanner  the scanner that needs to be started
```

```
   */

  public function startScannerTimer():void { //scanner:String):void {

//!REMOVE  :
  //gameData.incrementItemCountInInventory(1);
  //gameData.incrementItemCountInInventory(3);

    if ((gameData.getItemCountInInventory(3) > 0)
    &&
    (gameData.getItemCountInInventory(1) > 0)) {//scanner 2 (index 3 of items
    panel)
    //start timer for the scanner2
    scanner2Timer.start();
    scanner2Timer.addEventListener(TimerEvent.TIMER, scannerTimerHandler);

    //stop the other scanner (scanner1)
    scanner1Timer.stop();
    scanner1Timer.removeEventListener(TimerEvent.TIMER, scannerTimerHandler);

//    trace("startScanner2Timer");
    }else if ((gameData.getItemCountInInventory(1) > 0)
    &&
    (gameData.getItemCountInInventory(3) <= 0)) {//scanner 1 (index 1 of items
    panel)
    //start timer for the scanner1
    scanner1Timer.start();
    scanner1Timer.addEventListener(TimerEvent.TIMER, scannerTimerHandler);

    //stop the other scanner (scanner2)
    scanner2Timer.stop();
    scanner2Timer.removeEventListener(TimerEvent.TIMER, scannerTimerHandler);

//    trace("startScanner1Timer");
    }
  }

  /**
   * Handles the timers, executing the necessary events, when the
   * timers finish (and resetting the timers, when they're done).
   * @param  e  TimerEvent
   */
  public function scannerTimerHandler(e:TimerEvent):void {
    //trace("event triggered by: " + e.currentTarget);
    var indx:int;
    var scann:String;
    if (e.currentTarget == scanner1Timer) {
    indx = gameData.getScannerCurrentIndex("1");
    scann = "1";
//    trace("TIMER1: \t current time: " + getTimer());
    }else if (e.currentTarget == scanner2Timer) {
    indx = gameData.getScannerCurrentIndex("2");
    scann = "2";
//    trace("TIMER2: \t current time: " + getTimer());
    }
```

```
//!CHANGE SCANNER!

    if (area3[indx][0]){ //if a person exists in that box
    if (!area3[indx][0].getAdmin()) {
      //if the person in the box is not green, dispatch event to show box.
      tops[indx].dispatchEvent(new MouseEvent(MouseEvent.CLICK));
    }
    }else {//no person in box
    tops[indx].dispatchEvent(new MouseEvent(MouseEvent.CLICK));
//    trace("dispatch");

    }
//trace("count scan 1: " + gameData.getItemCountInInventory(1) + "\tcount scan 2:
" +
//gameData.getItemCountInInventory(3));
//!
    //trace("just looked at index: " + (gameData.getScannerCurrentIndex("2")));
    gameData.incrementScannerCurrentIndex(scann);//increment the scanner to check
    the next box
//trace(gameData.getScannerCurrentIndex("1") + " " +
gameData.getScannerCurrentIndex("3"));
  }



  /**
   * Make NAC work
   */
  public function runNAC():void {


//!REMOVE
    //gameData.incrementItemCountInInventory(5);
//!make objects green??????

    //check that game has not been paused:
    if (gameData.getGameMode() == "play") {
    //check that NAC has been bought
    if (gameData.getItemCountInInventory(5) > 0) {

      //if the box tops are still there, remove them
      if (tops[0].alpha != 0) {
      for (var i:int = 0; i < tops.length; i++) {
        tops[i].visible = false;
      }
      }

      //removes objects that are unauthorized
      for (var i:int = 0; i < area3.length; i++) {
      if (area3[i][0]) {
        if (!area3[i][0].getAdmin()) {
        area3[i][0].mouseEnabled = false;//remove mouse overs
        }
```

```
        if ((area3[i][0].mood != 1) && (!area3[i][0].getAdmin())) {//if the
        person in the box is not green, dispatch event to show box.
        tops[i].dispatchEvent(new MouseEvent(MouseEvent.CLICK));//show contents
        of box
        tops[i].dispatchEvent(new MouseEvent(MouseEvent.CLICK));//remove contents
        (since not
        }

      }
      }
    }
    }
  }

  /**
   * Adds the separators (which will be used to
   * find out where to place walls) to the stage.
   */
  public function addWalls():void {
    //add separationOutlines
    miscContainer.addChild(wallV);
    miscContainer.addChild(wallH);

    //store the separator outlines into an array
    wallArray.push(wallV);
    wallArray.push(wallH);
  }

  /**
   * Show walls, if they have already been bought.
   */
  public function walls():void {
    //if wall1 (vertical) has been bought, then put it into use
    if (gameData.getItemCountInInventory(2) > 0) {
    //change the height of the wall, based on the current level
    wallV.changeHeight(gameData.getLevel());
    wallV.visible = true;
    }
    //if wall2 (horizontal) has been bought, then put it into use
    if (gameData.getItemCountInInventory(4) > 0) {
    wallH.visible = true;
    }
  }

//! needs to only be checked only when level starts  - once
  /**
   * Returns the area of the address space to be checked, based on
   * the current level and what walls have been bought.
   * @return  Array of indeces that need to be checked in area3
   */
  public function wallAreaToBeChecked():Array {
    var data:Array = new Array();//array to store the indeces of the area to be
    checked

//!testing
```

```
//gameData.setLevel(testLevel);
/*
gameData.incrementItemCountInInventory(2);
wallV.visible = true;
gameData.incrementItemCountInInventory(4);
wallH.visible = true;
*/


    if (gameData.getLevel() == 4) {//we have 25 squares
    if (gameData.getItemCountInInventory(2) > 0) {//wall V has been bought
      //two zones
      data.push(2);//num of zones
      data.push([0, 1, 2, 5, 6, 7, 10, 11, 12, 15, 16, 17, 20, 21, 22]);
      data.push([3, 4, 8, 9, 13, 14, 18, 19, 23, 24]);
    }else {//wall V has not been bought
      //one zone
      data.push(1);//num of zones
      var temp:Array = new Array();
      for (var i:int = 0; i < 25; i++) {
      data.push(i);
      }
    }
    }else if (gameData.getLevel() > 4) {//we have 30 squares
    if ((gameData.getItemCountInInventory(2) > 0) &&
    (gameData.getItemCountInInventory(4) > 0)) {//bought both walls
      //four zones in total
      data.push(4);//num of zones
      data.push([0, 1, 2, 5, 6, 7, 10, 11, 12]);
      data.push([3, 4, 8, 9, 13, 14]);
      data.push([15, 16, 17, 20, 21, 22, 25, 26, 27]);
      data.push([18, 19, 23, 24, 28, 29]);
    }else if (gameData.getItemCountInInventory(4) > 0) {//wall H has been bought
      //two zones
      data.push(2);//num of zones
      var temp:Array = new Array();
      for (var i:int = 0; i < 14; i++) {
      temp.push(i);
      }
      data.push(temp);

      temp = new Array();
      for (var i:int = 15; i < 29; i++) {
      temp.push(i);
      }
      data.push(temp);
    }else if (gameData.getItemCountInInventory(2) > 0) {//wall V has been bought
      //two zones
      data.push(2);//num of zones
      data.push([0, 1, 2, 5, 6, 7, 10, 11, 12, 15, 16, 17, 20, 21, 22, 25, 26,
      27]);
      data.push([3, 4, 8, 9, 13, 14, 18, 19, 23, 24, 28, 29]);
    }else if ((gameData.getItemCountInInventory(2) <= 0) &&
    (gameData.getItemCountInInventory(4) <= 0)) {
```

```
      //single zone
      data.push(1);//num of zones
      var temp:Array = new Array();
      for (var i:int = 0; i < 30; i++) {
      data.push(i);
      }
    }
    }
//!remove - for testing

    //trace("level: " + gameData.getLevel());
    //trace(data.length);

    if (data[0] == 1) {
    for (var i:int = 1; i < data.length; i++) {
      //trace("data ["+i+"]: " + data[i]);
    }

    //trace("data 1: " + data[1].length);

    }else {
    for (var i:int = 1; i < data.length; i++) {
      //trace("i val: " + i);
      for (var j:int = 0; j < data[i].length; j++) {
      //trace("data ["+i+"]["+j+"]: " + data[i][j]);
      }
    }
    }


    return data;
  }

  /**
   * Adds all admins that are needed to area3.
   */
  public function addAdmin():void {
    var temp:Array = wallAreaToBeChecked();

    //check how many admins are needed
    var numAdminsToAdd:Array = new Array(temp[0]);//length=amount of admins needed

    if (temp[0] == 1) {
    numAdminsToAdd[0] = true;//need admin (will change, if need be)
    for (var i:int = 1; i < (temp.length - 1); i++) {
      if (area3[temp[i]][0] != null) {
      if (area3[temp[i]][0].getAdmin()) {//is an admin is taking up that space
        numAdminsToAdd[0] = false;//don't need that admin
      }
      }
    }

    //figure out where can add admin that is needed (if any)
    if (numAdminsToAdd[0]) {//if there is a need to add an admin
      var shortTemp:Array = new Array();//stores empty spaces in areas
```

```
      for (var i:int = 1; i < (temp.length - 1); i++) {
      //find out if admins are already in place
      if (!area3[temp[i]][1]) { //check if spot is empty
        shortTemp.push(temp[i]);
      }
      }
      var p:Person = new Person(true);
      peopleContainer.addChild(p);//add person to stage

      //find out where this admin will go
      p.movingToBox = temp[Math.floor(Math.random() * (shortTemp.length -
      1))];//random number to find out where to place admin
      //move obj to designated address
      p.x = addresses[p.movingToBox].coordX + 30;
      p.y = addresses[p.movingToBox].coordY + 30;

      //move obj to area3
      area3.splice(p.movingToBox, 1, [p, true]);
      area3[p.movingToBox][0].alpha = 1;//visible!
//!this section will screw up my listeners!!!!

      //hide boxtop at the index
      tops[p.movingToBox].visible = false;
      p.mouseEnabled = false;

      //make admin not clickable, if it is clickable
      tops[p.movingToBox].removeEventListener(MouseEvent.CLICK,
      deletePersonHandler);

    }

    }else {
    //trace("length: " + temp.length);
    for (var i:int = 1; i < temp.length; i++) {
      numAdminsToAdd[i - 1] = true;//need admin (will change, if need be)
      //find out if admins are already in place
      for (var j:int = 0; j < temp[i].length; j++) {
      //trace(temp[i][j]);
      //trace(area3[temp[i][j]][0]);
      if (area3[temp[i][j]][0] != null) {//taken up by person
        if (area3[temp[i][j]][0].getAdmin()) {//is an admin taking up that space
        numAdminsToAdd[i - 1] = false;//don't need that admin
        }
      }
      }
    }

    //figure out where can add admins that are needed
    for (var i:int = 1; i < temp.length; i++) {
      if (numAdminsToAdd[i - 1]) {//if there is a need to add an admin
      var shortTemp:Array = new Array();//stores empty spaces in areas
      //find out if admins are already in place
      for (var j:int = 0; j < temp[i].length; j++) {
        if (!area3[temp[i][j]][1]) { //check if spot is empty
        shortTemp.push(temp[i][j]);
```

```
        //trace("temp["+i+"]["+j+"]: "+temp[i][j]);
        }
      }

      var p:Person = new Person(true);
      peopleContainer.addChild(p);//add person to stage

      //find out where this admin will go
      p.movingToBox = temp[i][Math.floor(Math.random() *
      (shortTemp.length))];//random number to find out where to place admin
      if (gameData.getDebug()) {
        //trace("obj will move to: " + p.movingToBox);
      }

      //move obj to designated address
      p.x = addresses[p.movingToBox].coordX + 30;
      p.y = addresses[p.movingToBox].coordY + 30;
      //move obj to area3
      area3.splice(p.movingToBox, 1, [p, true]);
      area3[p.movingToBox][0].alpha = 1;//visible!

      //hide boxtop at the index
      tops[p.movingToBox].visible = false;
      p.mouseEnabled = false;
      tops[p.movingToBox].mouseEnabled = false;
      //make admin not clickable, if it is clickable
      tops[p.movingToBox].removeEventListener(MouseEvent.CLICK,
      deletePersonHandler);
      }
    }
    }
  }

  /**
   * Makes the admin removable, if the admin has been compromised.
   * @param  pers:  admin object
   */
  public function makeAdminRemovable(pers:Person):void {
    pers.mouseEnabled = true;
    tops[pers.movingToBox].mouseEnabled = true;
    tops[pers.movingToBox].addEventListener(MouseEvent.CLICK,
    deletePersonHandler);
    tops[pers.movingToBox].visible = true;
    tops[pers.movingToBox].alpha = 0;
  }

  public function spreadInfection():void {
//!testing show circles:
  /*
    if (tops[0].alpha != 0) {
    for (var i:int = 0; i < tops.length; i++) {
      tops[i].alpha = 0;
    }
    }
  */
```

```
    //scan area3
    var temp:Array = wallAreaToBeChecked();

    var compromisedPeople:Boolean = false;//are there compromised people in the
    subnet?
    var adminIndex:int = -1;//indices of where the admins are in area3
    var compromiserIndx:int = -1;//index of who is compromising admin

    if (temp[0] == 1) { //one subnet
    for (var i:int = 1; i < (temp.length - 1); i++) {
      if (area3[temp[i]][0] != null) {
      if (!area3[temp[i]][0].getAdmin()) {//is not an admin
        if ((area3[temp[i]][0].color == "red") || (area3[temp[i]][0].mood == 4))
        {//compromised
        compromisedPeople = true;
        compromiserIndx = temp[i];
        }
      }else {//it is an admin
        if (area3[temp[i]][0].color == "gray") {//has not been compromised
        adminIndex = temp[i];//store the admin's index in the array
        }
      }
      }
    }
//!remove later on:
    //compromisedPeople = true;

    //compromise the admin:
    if ((compromisedPeople) && (adminIndex != -1) && (area3[adminIndex][0].color
    != "red")) {
      area3[adminIndex][0].setInfectedBy(compromiserIndx,
      area3[compromiserIndx][0]);//sets up who compromised the admin
      //compromiser starts the timer to spread infection to admin
      /*
      area3[adminIndex][0].returnTimer().addEventListener(TimerEvent.TIMER,
      infectionTimerHandler);
      area3[adminIndex][0].returnTimer().start();
      */

//      trace("compromised!");
      area3[adminIndex][0].compromiseAdmin();
      shield();
      makeAdminRemovable(area3[adminIndex][0]);//make admin removable (clickable)

      //infect all objects in subnet
      for (var j:int = 0; j < (temp.length - 1); j++) {
      if (area3[temp[j]][0] != null) {//person is holding this spot
        if (!area3[temp[j]][0].getAdmin()) {//is not an admin
        if (area3[temp[j]][0].color != "red") {//person not compromised
          //compromise person
          changeColor(4, "mood", area3, temp[j]);//change the person's color to
          red
        }
        }
      }
```

```
      }
    }
    }else {
    for (var i:int = 1; i < temp.length; i++) {
      //reset variables:
      compromisedPeople = false;//are there compromised people in the subnet?
      adminIndex = -1;//indices of where the admins are in area3
      compromiserIndx = -1;//index of who is compromising admin

      for (var j:int = 0; j < temp[i].length; j++) {
      if (area3[temp[i][j]][0] != null) {//taken up by person
        if (!area3[temp[i][j]][0].getAdmin()) {//is not an admin
        if ((area3[temp[i][j]][0].color == "red") || (area3[temp[i][j]][0].mood
        == 4)) {//person is compromised
          compromisedPeople = true;
          compromiserIndx = temp[i][j];
        }
        }else {//is admin
        if (area3[temp[i][j]][0].color == "gray") {//has not been compromised
          adminIndex = temp[i][j];//store the admin's index in the array
        }
        }
      }
      }

      //compromise the admin:
      if ((compromisedPeople) && (adminIndex != -1) &&
      (area3[adminIndex][0].color != "red")) {
//!will probably not need next line
      area3[adminIndex][0].setInfectedBy(compromiserIndx,
      area3[compromiserIndx][0]);//sets up who compromised the admin
      //compromiser starts the timer to spread infection to admin

      /*
       * area3[adminIndex][0].returnTimer().addEventListener(TimerEvent.TIMER,
       infectionTimerHandler);
      area3[adminIndex][0].returnTimer().start();
      */

//      trace("compromised!");
      area3[adminIndex][0].compromiseAdmin();
      shield();
      makeAdminRemovable(area3[adminIndex][0]);//make admin removable

      //infect all objects in subnet
      for (var j:int = 0; j < temp[i].length; j++) {
        if (area3[temp[i][j]][0] != null) {//taken up by person
        if (!area3[temp[i][j]][0].getAdmin()) {//is not an admin
          if (area3[temp[i][j]][0].color != "red") {//person is not compromised
          //compromise person
          changeColor(4, "mood", area3, temp[i][j]);//change the person's color
          to red
          }
        }
        }
```

```
      }
      }
    }
    }
  }
//! remove
  public function infectionTimerHandler(e:TimerEvent):void {
//    trace("hello");
    e.currentTarget.removeEventListener(TimerEvent.TIMER, infectionTimerHandler);
    //find out from which admin the timer comes from
    for (var i:int = 0; i < area3.length; i++) {
    if (area3[i][0]) {
      if (area3[i][0].returnTimer() == e.currentTarget) {
      //check if compromiser is still in area3
      if (area3[area3[i][0].getInfectedBy()[0]][0] ==
      area3[i][0].getInfectedBy()[1]) {
        infectAllInSubnet();//i);//if it is, compromise everyone in subnet

      }
      }
    }
    }
  }

  /**
   * Compromises all devices in the subnet
   */
  public function infectAllInSubnet():void {//indx:int):void {
    var temp:Array = wallAreaToBeChecked();
    var compr:Boolean = false;//has admin been compromised?

    if (temp[0] == 1) { //one subnet
    for (var i:int = 1; i < (temp.length - 1); i++) {
      if (area3[temp[i]][0] != null) {
      if (area3[temp[i]][0].getAdmin()) {//is an admin
        if (area3[temp[i]][0].color == "red") {//has not been compromised
        compr = true;//admin has been compromised
        break;
        }
      }
      }
    }
    if (compr) {//if admin has been compromised, infect all objects in subnet
      for (var j:int = 0; j < (temp.length - 1); j++) {
      if (area3[temp[j]][0] != null) {//person is holding this spot
        if (!area3[temp[j]][0].getAdmin()) {//is not an admin
        if (area3[temp[j]][0].color != "red") {//person not compromised
          //compromise person
          changeColor(4, "mood", area3, temp[j]);//change the person's color to
          red
        }
        }
      }
      }
    }
```

```
    }else {
    for (var i:int = 1; i < temp.length; i++) {
      compr = false;//reset compromise variable

      for (var j:int = 0; j < temp[i].length; j++) {
      if (area3[temp[i][j]][0] != null) {//taken up by person
        if (area3[temp[i][j]][0].getAdmin()) {//is an admin
        if (area3[temp[i][j]][0].color == "red") {//admin is compromised
          compr = true;
          break;
        }
        }
      }
      }
      if (compr) {//if admin has been compromised, infect all objects in subnet
      for (var j:int = 0; j < temp[i].length; j++) {
        if (area3[temp[i][j]][0] != null) {//taken up by person
        if (!area3[temp[i][j]][0].getAdmin()) {//is not an admin
          if (area3[temp[i][j]][0].color != "red") {//person is not compromised
          //compromise person
          changeColor(4, "mood", area3, temp[i][j]);//change the person's color
          to red
          }
        }
        }
      }
      }
    }
    }
  }
  /**
   * Changes the color of all of the objects in area3
   * (used for NAC).
   */
  public function changeColorAll():void {
    //if NAC has been bought, or debug var is set to true
    if ((gameData.getItemCountInInventory(5) > 0) || (gameData.getDebug())) {
trace("debug: "+gameData.getDebug());
    //change the color of the objects in the boxes
    //to the color that they are supposed to show
    for (var i:int = 0; i < area3.length; i++) {
      //if there is an object in the box, change its mood & color
      if ((area3[i][0] != null) && area3[i][0]) {
      changeColor(area3[i][0].mood, "both", area3, i);
      }
    }
    }
  }

  /**
   * stuff to do to test the game more easily
   */
  public function testing():void {
```

```
    //if in debug/testing mode, then make all tops invisible
    if (gameData.getDebug()) {

    }

    /*
    gameData.setLevel(3);
    if (tops[0].alpha != 0) {
      for (var i:int = 0; i < tops.length; i++) {
      tops[i].alpha = 0;
      }
    }
    */
  }

//!to be used for testing game
  public function nextLevelListener(e:MouseEvent):void {

    switch(gameData.getLevel()) {
    case 1:
      gameData.setPoints(160);
      break;
    case 2:
      gameData.setPoints(190);
      break;
    case 3:
      gameData.setPoints(210);
      break;
    case 4:
      gameData.setPoints(230);
      break;
    case 5:
      gameData.setPoints(250);
      break;
    case 6:
      gameData.setPoints(270);
      break;
    }
    /*
    //gameData.setPoints(160);
    gameData.setLevel(gameData.getLevel() + 1);
    trace("nextLevel: " + gameData.getLevel());
    startGameLevel(gameData.getPoints()+10, gameData.getLevel());
    */
  }

  /**
   * Returns the index of the admin, based on where the currently
   * compromised person is in the address space and how many sections
   * the address space has been subdivided into.
   * @param  ind:  index of infected person in address space
   * @return  the index of the admin in the same subnet
   */
  public function returnAdminIndexForPerson(ind:int):int {
//    trace("ind value: " + ind);
```

```
    //scan area3
    var temp:Array = wallAreaToBeChecked();
    var adminIndex:int = -1;//index of where the admin is in area3

    if (temp[0] == 1) { //one subnet
//    trace("temp.indexOf(ind): " + temp.indexOf(ind));
    if (temp.indexOf(ind) > 0) {//if person could be in area3
      for (var i:int = 1; i < (temp.length - 1); i++) {
//      trace(temp[i]);
      if (area3[temp[i]][0] != null) {//person is there
        if (area3[temp[i]][0].getAdmin()) {//it is an admin
        adminIndex = temp[i];//store the admin's index
//        trace("admin index: " + temp[i]);
        }
      }
      }
    }
    }else {//multiple subnets
    adminIndex = -1;//index of where the admin is in area3

    for (var i:int = 1; i < temp.length; i++) {
      if (temp[i].indexOf(ind) > 0) {//if person could be in this section of area3
      for (var j:int = 0; j < temp[i].length; j++) {
        if (area3[temp[i][j]][0] != null) {//taken up by person
        if (area3[temp[i][j]][0].getAdmin()) {//is an admin
          adminIndex = temp[i][j];//store the admin's index
        }
        }
      }
      }
    }
    }
//    trace("admin index: " + adminIndex);
    return adminIndex;
  }


  /**
   * Finds out all of the admin indeces in the whole address space
   * and returns them.
   * @return  an array of all of the admins' indeces
   */
  public function returnAdminIndeces():Array {
    //scan area3
    var temp:Array = wallAreaToBeChecked();
    var adminIndeces:Array = new Array();//indices of where the admins are in
    area3

    if (temp[0] == 1) { //one subnet
    for (var i:int = 1; i < (temp.length - 1); i++) {
//      trace(temp[i]);
      if (area3[temp[i]][0] != null) {//person is there
      if (area3[temp[i]][0].getAdmin()) {//it is an admin
        adminIndeces.push(temp[i]);//store the admin's index in the array
      }
```

```
      }
    }
    }else {//multiple subnets
    adminIndeces = new Array();//reset index of where the admins are in area3

    for (var i:int = 1; i < temp.length; i++) {
      for (var j:int = 0; j < temp[i].length; j++) {
      if (area3[temp[i][j]][0] != null) {//taken up by person
        if (area3[temp[i][j]][0].getAdmin()) {//is an admin
        adminIndeces.push(temp[i][j]);//store the admin's index
        }
      }
      }
    }
    }

    return adminIndeces;
  }

  /**
   * Shows the shield icon for one or more admins, if
   * another admin (not in the same address space) has been compromised.
   */
  public function shield():void {
    var admins:Array = returnAdminIndeces();//store the indeces of the admins in
    an array

    //go through the admin array and check for uncompromised admins
    for (var i:int = 0; i < admins.length; i++) {
    if ((area3[admins[i]][0].color != "red") && (area3[admins[i]][0].currentFrame
    != 4)){
      area3[admins[i]][0].gotoAndPlay(4);//make admin show shield

    }
    }
  }

  /**
   * Pauses the items from the items panel, preventing them from being
   * used and running, when the game is not in play mode.
   */
  public function pauseGameItems(mode:String):void {
    var paused:Boolean = (mode != "play");
//having trouble pausing hourglass... can only stop it and restart it...
    /*
    if (gameData.getHourglassTimer()) {//if hourglass timer exists
    if (paused) {//game is paused
      gameData.getHourglassTimer().stop();
      trace("hourglass running: " + gameData.getHourglassTimer().running);
    }else {
      //timer has been stopped (presumably from game being paused)
//need to check if timer has sec left over on it, and if so, then unpause it
      if (!gameData.getHourglassTimer().running) {
      gameData.getHourglassTimer().start();
      trace("hourglass running: " + gameData.getHourglassTimer().running);
```

```
      }
    }
    }*/

    //pause the scanner items, if they have already been bought
    //check if the bronze scanner has been bought
    if ((gameData.getItemCountInInventory(3) > 0)){// &&
    (gameData.getItemCountInInventory(1) > 0)) {
    if (paused) {
      scanner2Timer.stop();//pause its timer
trace("scanner 2 stopped");
    }else {
      scanner2Timer.start();//start its timer
trace("scanner 2 started");
    }

    //check if the silver scanner has been bought
    }else if (gameData.getItemCountInInventory(1) > 0) {
    if (paused) {
      scanner1Timer.stop();//pause its timer
trace("scanner 1 stopped");
    }else {
      scanner1Timer.start();//start its timer
trace("scanner 1 started");
    }
    }
  }

  /**
   * Sets up the messages panel that shows up at the
   * end of every level.
   */
/*
  public function setUpMessages():void {
    miscContainer.addChild(messages);
  }
*/
  /**
   * Resets variables.
   */
  public function resetAll():void {
    removeAllObjects();
    container = new Sprite();
    peopleContainer = new Sprite();
    boxTopsContainer = new Sprite();
    miscContainer = new Sprite();
    form = new Form(); //form that the person needs to fill out
    table = new Table();//desk where player sits
    lineSeg = new MovieClip(); //line on left-hand side of address boxes
    hourglass = new Hourglass();//hourglass object that contains num of
    hourglasses in inv
    timeForm = new Timer(100, 20);//!!!5, 1);
    deltaTime = 10;// 5; //time it takes unauthorized circles to get to boxes
    addresses = new Array(); //addresses for the people to go to
    tops = new Array(); //contains box tops
```

```
    people = new Array(); //list of people/devices
    area3 = new Array(); //all of the devices that are in the boxes
    area2 = new Array();
    gameOverSplash = new GameOverScreen();//game over screen
    gameOverContainer = new Sprite();
    levelCompleteSplash = new LevelCompleteScreen();
    levelCompleteContainer = new Sprite();
    gameCompleteSplash = new YouWin();
    gameCompleteContainer = new Sprite();
    pointsText = new TextField();
    pointsTextFormat = new TextFormat();//formatting
    levelText = new TextField();
    levelTextFormat = new TextFormat();//formatting
    pointsTextScreen = new TextField();
    pointsTextScreenFormat = new TextFormat();//formatting
    gamePauseText = new GamePauseText();
    hourglassInventoryText = new TextField();
    hourglassInventoryTextFormat = new TextFormat();

    scanner1Timer.removeEventListener(TimerEvent.TIMER, scannerTimerHandler);
    scanner2Timer.removeEventListener(TimerEvent.TIMER, scannerTimerHandler);

    scanner1Timer = new Timer(1000, 0);//scanner1 checks one box every 1 sec
    scanner2Timer = new Timer(500/2, 0);//scanner2 checks one box every 0.25 sec
    midpointBoxes = new Array();
    screen = new Screen();//cover to make mouse actions unavailable
    click = false; //no click
    gameData = new GameData(screen);//, getMainInstance()); //points and current
    level of the game
    itemsPanel = new ItemsPanel(gameData);
    itemsTab = new ItemsTab();
    currentPerson = new Person(false);
    freeSpotsCount = 0;//number of free spots
    numLeaveLine = 0;
    awesome = 0;//for testing purposes
    temp = getTimer();
    wallH = new SepOutline(400, 230, 0);
    wallV = new SepOutline(579, 50, 90, (1 + (1 / 5)));
    wallArray = new Array();//will contain the two sepOutlines
    wallArea3 = new Array();//stores the different zones created from the walls
    testLevel = 5;
    nextLevel = new NextLevel();

    messages = new Messages();

    //scanner1Timer.stop();
    //scanner2Timer.stop();

trace("resetAll");
  }
  }
}
```

Box.as

```
package classes {

  import flash.display.MovieClip;


  /**
   * Creates a Box object with all of its properties.
   * @author Era Vuksani
   */

  public class Box extends MovieClip {

  public var coordX:Number;
  public var coordY:Number;

  /**
   * Initializes the x and y coords for Box.
   * @param  X:   x coord
   * @param  Y:   y coord
   */
  public function init(X:Number, Y:Number):void{
    //x and y coords input
    this.x = X;
    this.y = Y;
  }

  /**
   * Updates the coordX and coordY properties of the BoxTop.
   *
   * @param  X:   x coord
   * @param  Y:   y coord
   */
  public function update(X:Number, Y:Number):void{
    //properties
    coordX = X;
    coordY = Y;
  }
  }
}
```

BoxTop.as

```
package classes{
  import flash.display.MovieClip;
  import fl.transitions.Tween;
  import fl.transitions.easing.*;
  import fl.transitions.*;

  /**
   * Creates a BoxTop object with all of its properties.
```

```
   * @author Era Vuksani
   */
  public class BoxTop extends MovieClip {

  public var coordX:Number;
  public var coordY:Number;
  //private var removable:Boolean=true;

  /**
   * Initializes the x and y coords for the BoxTop.
   *
   * @param  X:   x coord
   * @param  Y:   y coord
   */
  public function init(X:Number, Y:Number):void{
    //x and y coords input
    this.x = X;
    this.y = Y;

//!testing:
    //this.visible = false;
  }

  /**
   * Updates the coordX and coordY properties of the BoxTop.
   *
   * @param  X:   x coord
   * @param  Y:   y coord
   */
  public function update(X:Number, Y:Number):void{
    //properties
    coordX = X;
    coordY = Y;
  }

  /**
   * Sets the removable variable.
   * @param  b  Boolean
   */
  /*
  public function setRemovable(b:Boolean):void {
    removable = b;
  }

  /**
   * Get the removable variable.
   */
  /*
  public function getRemovable():void {
    return removable;
  }
  */

  }
}
```

GameData.as

```
package classes {
  import flash.display.MovieClip;
  import flash.display.Sprite;
  import flash.events.FullScreenEvent;
  import flash.events.TimerEvent;
  import flash.utils.Timer;
  import flash.events.*;

  import classes.Wall;


  public class GameData {
  public static var gamePoints:Number;
  private static var gameLevel:Number;
  private static var gameMode:String;
  private static var itemInventory:Array = new Array();
  private static var debug:Boolean;
  //MAKE PRIVATE
  public static var screen:MovieClip;

//  private var timerArray:Array = new Array();//might not uses

  private static var hourglassTimer:Timer = new Timer(5000, 0);//manages how long
  the hourglass has until it expires.
  //5 seconds of speed set at 5px
  //timers for the scanners
  //private var scanner1Timer:Timer = new Timer(200, 0);//scanner1 checks one box
  every 10 sec
  //private var scanner2Timer:Timer = new Timer(5000, 0);//scanner2 checks one
  box every 5 sec
  //private var testTimer:Timer = new Timer(10000, 0);//to remove

  //should be 10
  private static var speedOfLine:int = 0;//the initial speed of the movement of
  the people on the line (initially = 10)
  private static var speedOfLineOriginal:int = 0;//the previously recorded speed
  of the movement of the people on the line (initially = 10)

  //scanner data:
  private static var scan1CurrentIndex:int = 0;
  private static var scan2CurrentIndex:int = 0;

  //NAC data:
  //private static var NACCurrentIndex:int = 0;

  private static var numVisibleBoxes:int = 0;//number of visible boxes

  //resulting areas, based on the walls
  /*
  private static var wallArea1:Array = new Array();//1st area to the top left or
  whole area, if no wall
  private static var wallArea2:Array = new Array();//topmost right area
  private static var wallArea3:Array = new Array();//bottom left area
```

```
private static var wallArea4:Array = new Array();//bottom right area
*/

public function GameData(scr:MovieClip) {
  debug = false;

  resetAll();//reset all vars
  gamePoints = 100;
  gameLevel = 1;
///  gameMode = "play";
  setSpeedOfLineBasedOnLevel();//reset the speed of the line
  screen = scr;
  if (debug) {
  //setLevel(7);
  }

}

/** Debugging code
 */
public function print(str:String):void {
  if (debug){
  //trace(str);
  }
}
//points-related functions

/**
 * Sets the game points to the current score.
 * @param  pts
 */
public function setPoints(pts:Number):void {
  gamePoints = pts;
  print("points= " + String(gamePoints));
  //print("array of items: "+String(itemInventory[0]));
}

/**
 * Gets the current score.
 * @return the current number of points
 */
public function getPoints():Number {
  return gamePoints;
}

/**
 * Set the speed of the line, based on the current
 * level of the game.
 */
public function setSpeedOfLineBasedOnLevel():void {
  switch(gameLevel) {
  case 1:
    setSpeedOfLine(10);
    break;
```

```
case 2:
  setSpeedOfLine(15);
  break;

case 3:
  setSpeedOfLine(20);
  break;

case 4:
  setSpeedOfLine(20);
  break;

case 5:
  setSpeedOfLine(30);
  break;

case 6:
  setSpeedOfLine(30);
  break;

case 7:
  setSpeedOfLine(100);
  break;
}
/*
 * switch(gameLevel) {
case 1:
  setSpeedOfLine(3);
  break;

case 2:
  setSpeedOfLine(15);
  break;

case 3:
  setSpeedOfLine(20);
  break;

case 4:
  setSpeedOfLine(20);
  break;

case 5:
  setSpeedOfLine(30);
  break;

case 6:
  setSpeedOfLine(30);
  break;

case 7:
  setSpeedOfLine(100);
  break;
}
*/
```

```
  speedOfLineOriginal = speedOfLine;
}

/**
 * Sets the speed of the movement of the people on the line.
 * @param  num
 */
public function setSpeedOfLine(num:int):void {
  speedOfLine = num;
//  speedOfLineOriginal = num;
}

/**
 * Returns the speed of the line.
 * @return   speed of line
 */
public function getSpeedOfLine():int{
  return speedOfLine;
}

/**
 * Sets the gameMode variable to be the same as the Main class'.
 * @param  str (gameMode)
 */
public function setGameMode(str:String):void {
  gameMode = str;
}

/**
 * Returns the current gameMode status.
 * @return gameMode
 */
public function getGameMode():String {
  return gameMode;
}

//level-related function

/**
 * Sets the game level.
 * @param  level
 */
public function setLevel(level:Number):void {
  gameLevel = level;
}

/**
 * Gets the game level.
 * @return the game level
 */
public function getLevel():Number {
  return gameLevel;
}

/**
```

```
 * Returns the value of the debug (whether the game is being debugged or not).
 * @return debug
 */
public function getDebug():Boolean {
  return debug;
}

/**
 * Push an object into the game's item inventory.
 * @param  item
 */
public function pushItemInInventory(item:Array):void {
  itemInventory.push(item);
}



/**
 * Get the requested item's count from the inventory.
 * @param  indx
 * @return   the count of the item in the inventory
 */
public function getItemCountInInventory(indx:Number):Number {
  return itemInventory[indx][0];
}

/**
 * Decrement the hourglass' inventory.
 */
public function decrementItemCountInInventory():void {
  itemInventory[0][0]--;
}

/**
 * Get the requested item's count from the inventory and increment it or change
 * it to the input value.
 * @param  indx
 * @param  ... increm
 * @return   the count of the item in the inventory
 */
public function incrementItemCountInInventory(indx:Number, ... increm):void {
  if (increm.length == 0) {
  itemInventory[indx][0]++;
  //print("incrementation of " + indx + " = " + itemInventory[indx][0]);
  }else {
  itemInventory[indx][0] += increm[0];
  }
}

/**
 * Get the specified item from the inventory.
 * @param  indx
 * @return  item
 */
public function getItem(indx:Number):MovieClip {
  return itemInventory[indx][1];
```

```
}

/**
 * Get the cost of the desired item.
 * @param  indx
 * @return   cost of the item
 */
public function getItemCost(indx:Number):Number {
  print("cost of item is: " + itemInventory[indx][2]);
  return itemInventory[indx][2];
}


/**
 * Hides or shows screen, depending on boolean provided.
 * @param  hide: Boolean
 */

public function setHideScreen(hide:Boolean):void {
  screen.visible = (!hide);
  /*
  //if hide is true, set screen to invisible
  if (hide) {
  //screen.x = 1100;
  screen.visible = false;
  print("screen is invisi");

  //if hide is false, set screen to visible
  }else {
  //screen.x = 0;
  screen.visible = true;
  print("screen is visi");
  }
  */
}

/**
 * Slows down the speed of the line, based on whether or not the
 * player has at least one hourglass in his inventory.
 */
public function slowDownLineSpeed():void {
  startHourglassTimer();
}

/**
 * Stars the hourglass' timer.
 */
public function startHourglassTimer():void {
  //hourglassTimer.reset();
  hourglassTimer.start();
  setSpeedOfLine(5);

  hourglassTimer.addEventListener(TimerEvent.TIMER, hourglassTimerHandler);
}
```

```
  /**
   * Takes care of removing the event listener for the hourglass,
   * when the timer is done.
   * @param  e  TimerEvent
   */
  public function hourglassTimerHandler(e:TimerEvent):void {
    //if timer has stopped, remove the eventhandler and return the speed of the
    line to the normal one (10px)
    setSpeedOfLine(speedOfLineOriginal);
//    trace("previous speed of line: "+speedOfLineOriginal);
    hourglassTimer = new Timer(5000, 0);
    hourglassTimer.removeEventListener(TimerEvent.TIMER, hourglassTimerHandler);
  }

  /**
   * Returns the hourglass timer (for use in Main class).
   * @return  hourglassTimer.
   */
  public function getHourglassTimer():Timer {
    return hourglassTimer;
  }

  /**
   * Return the scanner's current index (ie: current box
   * that the scanner is scanning).
   * @param  scanner:  the scanner's name
   * @return  the index of the scanner
   */
  public function getScannerCurrentIndex(scanner:String):int {
    var i:int = -1;
    if ((scanner == "scanner1") || (scanner == "1")){
    i = scan1CurrentIndex;
    }else if ((scanner == "scanner2") || (scanner == "2")){
    i = scan2CurrentIndex;
    }
    /*else if (scanner == "NAC") {
    i = NACCurrentIndex;
    }*/

    return i;
  }

  /**
   * Resetes the scanners' index to be 0.
   */
  public function resetScannerCurrentIndex():void {
    scan1CurrentIndex = 0;
    scan2CurrentIndex = 0;
  }

  /**
   * Increment the current index of the specified scanner.
   * @param  scanner: The scanner's name or number
   */
  public function incrementScannerCurrentIndex(scanner:String):void {
```

```
    var i:int = -1;
//!add if game is not paused
    //find out which scanner to increment the index of
    if ((scanner == "scanner1") || (scanner == "1")) {
    //if the index is getting out of the max number of boxes that the game
    contains,
    //then return the incremented index
    if (scan1CurrentIndex < (numVisibleBoxes - 1)) {
      scan1CurrentIndex++;
    //otherwise, return the index back down to 0
    }else {
      scan1CurrentIndex = 0;
    }

    }else if ((scanner == "scanner2") || (scanner == "2")) {
    //if the index is getting out of the max number of boxes that the game
    contains,
    //then return the incremented index
    if (scan2CurrentIndex < (numVisibleBoxes - 1)) {
      scan2CurrentIndex++;
    //otherwise, return the index back down to 0
    }else {
      scan2CurrentIndex = 0;
    }

    }

  }

  /**
   * Sets the number of visible boxes.
   * @param  n  number of visible boxes
   */
  public function setNumVisibleBoxes(n:int):void {
    numVisibleBoxes = n;
  }

  /**
   * Returns the number of visible boxes.
   * @return  numVisibleBoxes
   */
  public function getNumVisibleBoxes():int {
    return numVisibleBoxes;
  }

  /**
   * Reset all variables; useful for restarting game.
   */
  private function resetAll():void {
    itemInventory = new Array();
    hourglassTimer = new Timer(5000, 0);//manages how long the hourglass has
    until it expires.
    speedOfLine = 0;//the initial speed of the movement of the people on the line
    (initially = 10)
    speedOfLineOriginal = 0;//the previously recorded speed of the movement of
```

```
    the people on the line (initially = 10)
    scan1CurrentIndex = 0;
    scan2CurrentIndex = 0;
  }
  }
}
```

ItemsPanel.as

```
package classes {


  import flash.display.MovieClip;
  import flash.display.Sprite;
  import flash.utils.Timer;
  import flash.utils.getTimer;
  import flash.utils.*;

  import fl.transitions.*;
  import fl.transitions.easing.*;

  import flash.text.*;
  import fl.controls.TextInput;

  import flash.events.MouseEvent;
  import flash.events.Event;

  import classes.GameData;

  import MouseOverItems;

  public dynamic class ItemsPanel extends MovieClip {
  //Containers:
  //contains all items and other containers
  public static var backgroundContainer:Sprite = new Sprite();
  //container for first item objects
  private static var firstItemContainer:Sprite = new Sprite();
  //container for second item objects
  private static var secondItemContainer:Sprite = new Sprite();
  //container for third item objects
  private static var thirdItemContainer:Sprite = new Sprite();
  //container for tabContainer objects
  //private static var tabContainer:Sprite = new Sprite();

  //items:

  //items for background container:
  private static var itemsPanelOutline:ItemsPanelOutline = new ItemsPanelOutline();
  //private var pageArrowRight:PageArrowRight = new PageArrowRight();
  //private var pageArrowLeft:PageArrowLeft = new PageArrowLeft();

  //private static var pageNumber:TextField = new TextField();
```

```
//items for tabContainer:
//public var tab:ItemsTab = new ItemsTab();//items tab
//private var arrowLeft:ArrowLeft = new ArrowLeft();
//private var arrowRight:ArrowRight = new ArrowRight();
//private var itemsText:ItemsText = new ItemsText();//text above arrows


//Items container:

//page 1 of items container
//items for the 1st item container
private static var itemFirst:ItemFirst = new ItemFirst();
private static var buyButtonFirst:BuyButton = new BuyButton();

//items for the 2nd item container
private static var itemSecond:ItemSecond = new ItemSecond();
private static var buyButtonSecond:BuyButton = new BuyButton();

//items for the 3rd item container
private static var itemThird:ItemThird = new ItemThird();
private static var buyButtonThird:BuyButton = new BuyButton();


//page 2 of items container
//items for the 4th item container
private static var itemFourth:ItemFourth = new ItemFourth();

//items for the 5th item container
private static var itemFifth:ItemFifth = new ItemFifth();

//items for the 6th item container
private static var itemSixth:ItemSixth = new ItemSixth();

//input text fields
//page number
      //private var pageNumber:TextInput = new TextInput();

//item 1:
      private static var quantityToBuyFirst:TextInput = new TextInput();
      private static var inventoryFirst:TextInput = new TextInput();

//item 2:
      //private var quantityToBuySecond:TextInput = new TextInput();
      //private var inventorySecond:TextInput = new TextInput();

//item 3:
      //private var quantityToBuyThird:TextInput = new TextInput();
      //private var inventoryThird:TextInput = new TextInput();

//private var currentPts:RemainingPts = new RemainingPts();
private static var currentPts:TextField = new TextField();

private static var textFormat:TextFormat = new TextFormat();//formatting
private static var currentPtsTextFormat:TextFormat = new
```

```
  TextFormat();//formatting for the current points text

  //text at top right of screen (current level)
  private static var levelText:TextField = new TextField();
  private static var levelTextFormat:TextFormat = new TextFormat();//formatting

  private static var gameData:GameData;
  private static var itemsArray:Array = new Array();
  private static var itemsPanelItems:Array = new Array();//all items of this
  class will be stored here
  private static var hidden:Boolean;

  //what appears when you mouse over items in items panel
  private static var upgradesContainer2:MouseOverItems = new MouseOverItems();
  private static var upgradesContainer3:MouseOverItems = new MouseOverItems();

//!remove:
  //testing vars:
  public var lvl:int = 5;

  /**
   * Constructor for items panel; sets up the tabContainer, and different
   * containers for the different items and their properties.
   */
  public function ItemsPanel(gd:GameData) {
    resetAll();//resets all vars
    gameData = gd;
    hidden = true;//panel is initially hidden
    //setUpTabContainer();
    setUpFirstItemContainer();
    setUpSecondItemContainer();
    setUpThirdItemContainer();
    setUpBackgroundContainer();
    setUpTextFormat();
    setUpArrays();

    //items related functions
    setUpItems();//set up 3 first items
    addItemListeners();//upgrades related

    /*
    if (gameData.getDebug()) {
    trace("level: " + gameData.getLevel());
    }
    */
    displayInventory();
    //changeAll();


    //trace("1's current frame#3: " + itemsArray[1][0].currentFrame);
  }

  /**
   * Add listeners to the item images to show what the
   * items can be upgraded to, later on in the game.
```

```
 */
public function addItemListeners():void {
  itemSecond.addEventListener(MouseEvent.MOUSE_OVER, itemsRollOverListener);
  itemThird.addEventListener(MouseEvent.MOUSE_OVER, itemsRollOverListener);
  itemFourth.addEventListener(MouseEvent.MOUSE_OVER, itemsRollOverListener);

  itemSecond.addEventListener(MouseEvent.MOUSE_OUT, itemsRollOutListener);
  itemThird.addEventListener(MouseEvent.MOUSE_OUT, itemsRollOutListener);
  itemFourth.addEventListener(MouseEvent.MOUSE_OUT, itemsRollOutListener);
}

/**
 * Shows you what you can upgrade the rolled over item to.
 * @param  e:   MouseEvent
 */
public function itemsRollOverListener(e:MouseEvent):void {
  if ((e.currentTarget == itemSecond) && itemSecond.visible) {//target is
  silver scanner
  upgradesContainer2.gotoAndStop(1);
  upgradesContainer2.visible = true;

  }else if ((e.currentTarget == itemThird) && itemThird.visible) {//target is
  vertical wall
  upgradesContainer3.gotoAndStop(3);
  upgradesContainer3.visible = true;

  }else if ((e.currentTarget == itemFourth) && itemFourth.visible) {//target is
  bronze scanner
  upgradesContainer2.gotoAndStop(2);
  upgradesContainer2.visible = true;

  }

}

/**
 * Hides you what you can upgrade the rolled over item to.
 * @param  e:   MouseEvent
 */
public function itemsRollOutListener(e:MouseEvent):void {
  //target is silver scanner or bronze scanner
  if ((e.currentTarget == itemSecond) || (e.currentTarget == itemFourth)) {
  upgradesContainer2.visible = false;

  }else if (e.currentTarget == itemThird) {//target is vertical wall
  upgradesContainer3.visible = false;

  }
}

public function setUpArrays():void {
  //add items only to panel
  itemsArray =
  [[itemFirst, buyButtonFirst],
  [itemSecond, buyButtonSecond],
```

```
    [itemThird, buyButtonThird],
    [itemFourth, buyButtonSecond],
    [itemFifth, buyButtonThird],
    [itemSixth, buyButtonSecond]];


    //push the items into the game's inventory
    //first index = number of items currently in the inventory
    //second index = name of item
    //third index = cost of one item
//!uncomment

    gameData.pushItemInInventory([0, itemFirst, 10]);//hourglass - stays
    gameData.pushItemInInventory([0, itemSecond, 50]);//silver scanner
    gameData.pushItemInInventory([0, itemThird, 20]);//vertical wall
    gameData.pushItemInInventory([0, itemFourth, 100]);//bronze scanner
    gameData.pushItemInInventory([0, itemFifth, 40]);//horizontal wall
    gameData.pushItemInInventory([0, itemSixth, 150]);//NAC (gold scanner)


//!testing: remove
/*
    gameData.pushItemInInventory([0, itemFirst, 1]);//hourglass - stays
    gameData.pushItemInInventory([0, itemSecond, 1]);//silver scanner
    gameData.pushItemInInventory([0, itemThird, 1]);//vertical wall
    gameData.pushItemInInventory([0, itemFourth, 1]);//bronze scanner
    gameData.pushItemInInventory([0, itemFifth, 1]);//horizontal wall
    gameData.pushItemInInventory([0, itemSixth, 1]);//NAC (gold scanner)
*/
////////////////////////////////////////////////////////////////////////

    itemsPanelItems =
    [itemsPanelOutline,
    //pageArrowRight,
    //pageArrowLeft,
    itemFirst,
    buyButtonFirst,
    itemSecond,
    buyButtonSecond,
    itemThird,
    buyButtonThird,
    itemFourth,
    itemFifth,
    itemSixth,
    //pageNumber,
    quantityToBuyFirst,
    inventoryFirst];
    //quantityToBuySecond,
    //inventorySecond,
    //quantityToBuyThird,
    //inventoryThird];
  }

  /**
   * Sets up the formatting for all text boxes.
```

```
 */
private function setUpTextFormat():void {
  //font to embed
  var pointsFont:Courier = new Courier();

  //points text (topmost left) on game screen
  textFormat.font = pointsFont.fontName;

  //formatting:
  textFormat.color = 0x000000;
  textFormat.size = 12;


  //font to embed
  var pointsFont2:Courier = new Courier();
  currentPtsTextFormat.font = pointsFont2.fontName;

  //formatting:
  currentPtsTextFormat.color = 0x000000;
  currentPtsTextFormat.size = 21;

  //current points textbox
  currentPts.x = 263+135;
  currentPts.y = 553 + 17;
  currentPts.autoSize = TextFieldAutoSize.LEFT;
  currentPts.defaultTextFormat = currentPtsTextFormat;
  currentPts.text = "Remaining Points: " + gameData.getPoints();

  backgroundContainer.addChild(currentPts);
}

//Set up the different containers that will hold the items.

/**
 * Sets up the background container properties and adds
 * it to the stage.
 */
private function setUpBackgroundContainer():void {
  backgroundContainer.x = 0;
  backgroundContainer.y = -10;

  //sets up the tabContainer's location on the stage
  //tabContainer.x = 901.45;
  //tabContainer.y = 93.35;

  //add all other containers and objects needed to background container
  //itemsPanelOutline.pageNumber.x = 195.1+349.5;
  //itemsPanelOutline.pageNumber.y = 546;

  itemsPanelOutline.x = 349.5;
  itemsPanelOutline.y = 20;

  //arrows at bottom of panel

  //pageArrowLeft.x = 106.95+349.5; // 127.5;
```

```
    //pageArrowLeft.y = 523.75+20; // 524.7;
    //pageArrowRight.x = 247.4+349.5; //231.45;
    //pageArrowRight.y = 523.75 + 20; //524.95;

    //change the page based on which arrow has been pressed
    //pageArrowLeft.addEventListener(MouseEvent.CLICK, changePageLeft);
    //pageArrowRight.addEventListener(MouseEvent.CLICK, changePageRight);

    firstItemContainer.x = 391;
    firstItemContainer.y = 62;

    secondItemContainer.x = 391;
    secondItemContainer.y = 230;

    thirdItemContainer.x = 391;
    thirdItemContainer.y = 400;

    //page number
    /*
    pageNumber.x = 545;
    pageNumber.y = 545;
    pageNumber.width = 20;
    pageNumber.height = 16;

    pageNumber.editable = true;
    pageNumber.enabled = true;
    pageNumber.alwaysShowSelection = true;
    pageNumber.maxChars = 1;
    */

    //when something is typed into pageNumber it figures out what
    //that is and changes the page accordingly
    //pageNumber.addEventListener(Event.CHANGE, changePageOnEnter);


    //add all containers:
    backgroundContainer.addChild(itemsPanelOutline);
    backgroundContainer.addChild(firstItemContainer);

    backgroundContainer.addChild(secondItemContainer);
    backgroundContainer.addChild(thirdItemContainer);

    //backgroundContainer.addChild(tabContainer);
    //backgroundContainer.addChild(pageNumber);

    //add items to background container
    //backgroundContainer.addChild(pageArrowLeft);
    //backgroundContainer.addChild(pageArrowRight);



}

/**
```

```
   * Sets up the tabContainer that is used to move the panel.
   */
  /**
   *
   * private function setUpTabContainer():void {
    tab.x = 0;
    tab.y = 0;

    tabContainer.addChild(tab);


  }
  */

  /**
   * Sets up the first item in the list, which is Red Bull.
   * This includes name, description, cost, etc...
   */
  private function setUpFirstItemContainer():void {
    //1st item
    //coords for bg obj
    itemFirst.x = 0;// 160;
    itemFirst.y = 0;// -230;
    /*itemFirst.x = 41.8;
    itemFirst.y = 48.9;*/
//!    itemFirst.cost = 10;

    //4th item
    //coords for bg obj
/*
    itemFourth.x = 42.6;
    itemFourth.y = 40.1;
    itemFourth.cost = 100;
*/
    /*
    //7th item
    //coords for bg obj
    itemSeventh.x = 50.25;
    itemSeventh.y = 13.15;
    itemSeventh.cost = 150;
    */
    //item 1
    //set up quantity input box
    quantityToBuyFirst.x = 220;
    quantityToBuyFirst.y = 72;//104.05;
    quantityToBuyFirst.width = 16;

    quantityToBuyFirst.editable = true;
    quantityToBuyFirst.alwaysShowSelection = true;
    quantityToBuyFirst.maxChars = 2;
    quantityToBuyFirst.text = "1";


    //set up textbox containing quantity already in inventory
    inventoryFirst.x = quantityToBuyFirst.x;
```

```
    inventoryFirst.y = quantityToBuyFirst.y+25;//132.3;
    inventoryFirst.width = 16;

    inventoryFirst.editable = false;
    inventoryFirst.alwaysShowSelection = true;
    inventoryFirst.maxChars = 2;
    inventoryFirst.text = "0";

//!remove
    //buyButtonFirst.gotoAndPlay(2);


    //set up buy button
    buyButtonFirst.x = quantityToBuyFirst.x+70;
    buyButtonFirst.y = quantityToBuyFirst.y;
    buyButtonFirst.buttonMode = true;
    buyButtonFirst.addEventListener(MouseEvent.CLICK, buy);
    buyButtonFirst.addEventListener(MouseEvent.ROLL_OVER, rollOverListener);
    buyButtonFirst.addEventListener(MouseEvent.ROLL_OUT, rollOutListener);

    //add all items to firstItemContainer container
    firstItemContainer.addChild(itemFirst);
//!    firstItemContainer.addChild(itemFourth);
    //firstItemContainer.addChild(itemSeventh);

    firstItemContainer.addChild(quantityToBuyFirst);
    firstItemContainer.addChild(inventoryFirst);
    firstItemContainer.addChild(buyButtonFirst);
  }



  /**
   * Sets up the second item in the list, which is Address
   * Space Scanner.
   * This includes name, description, cost, etc...
   */
  private function setUpSecondItemContainer():void {
    //2nd item
    //coords for bg obj
    itemSecond.x = 0;
    itemSecond.y = 0;
//    itemSecond.cost = 50;

    //3rd item
    itemFourth.x = 0;
    itemFourth.y = 0;

    //5th item
    //coords for bg obj
    itemSixth.x = 0;
    itemSixth.y = 0;
//    itemFifth.cost = 40;

    //message showing upcoming upgrades of item
```

```
    upgradesContainer2.visible = false;
    upgradesContainer2.x = -310;
    upgradesContainer2.y = 0;

    //item 2
    //set up textbox containing quantity already in inventory
    /*
    inventorySecond.x = 253.15;
    inventorySecond.y = 275.8+11;
    inventorySecond.width = 14;

    inventorySecond.editable = false;
    inventorySecond.alwaysShowSelection = true;
    inventorySecond.maxChars = 2;
    */
    //set up buy button
    buyButtonSecond.x = quantityToBuyFirst.x+70;
    buyButtonSecond.y = 80;
    buyButtonSecond.buttonMode = true;
    buyButtonSecond.addEventListener(MouseEvent.CLICK, buy);
    buyButtonSecond.addEventListener(MouseEvent.ROLL_OVER, rollOverListener);
    buyButtonSecond.addEventListener(MouseEvent.ROLL_OUT, rollOutListener);

    //add all items to secondItemContainer container
    secondItemContainer.addChild(itemSecond);
    secondItemContainer.addChild(itemFourth);
    secondItemContainer.addChild(itemSixth);
    //secondItemContainer.addChild(inventorySecond);
    secondItemContainer.addChild(buyButtonSecond);
    secondItemContainer.addChild(upgradesContainer2);
  }

  /**
   * Sets up the third item in the list, which is NAC.
   * This includes name, description, cost, etc...
   */
  private function setUpThirdItemContainer():void {
    //3rd item
    //coords for bg obj
    itemThird.x = 0;
    itemThird.y = 0;
//    itemThird.cost = 20;

    //6th item
    //coords for bg obj
    itemFifth.x = 0;
    itemFifth.y = 0;
//    itemSixth.cost = 80;

    //item 3
    //set up textbox containing quantity already in inventory
    /*inventoryThird.x = 253.15;
    inventoryThird.y = 455.5;
    inventoryThird.width = 14;
```

```
    inventoryThird.editable = false;
    inventoryThird.alwaysShowSelection = true;
    inventoryThird.maxChars = 2;
    */

    //message showing upcoming upgrades of item
    upgradesContainer3.visible = false;
    upgradesContainer3.x = -310;
    upgradesContainer3.y = 0;

    //set up buy button
    buyButtonThird.x = quantityToBuyFirst.x+70;
    buyButtonThird.y = 80;
    buyButtonThird.buttonMode = true;
    buyButtonThird.addEventListener(MouseEvent.CLICK, buy);
    buyButtonThird.addEventListener(MouseEvent.ROLL_OVER, rollOverListener);
    buyButtonThird.addEventListener(MouseEvent.ROLL_OUT, rollOutListener);

    //add all items to thirdItemContainer container
    thirdItemContainer.addChild(itemThird);
    thirdItemContainer.addChild(itemFifth);
    //thirdItemContainer.addChild(inventoryThird);
    thirdItemContainer.addChild(buyButtonThird);
    thirdItemContainer.addChild(upgradesContainer3);
  }



  /*
  private function setUpTextBoxes():void {
    //first item
    quantityToBuyFirst.text = "1";
    inventoryFirst.text = "0";

    //second item
    //quantityToBuySecond.text = "0";
    //inventorySecond.text = "0";

    //third item
    //quantityToBuyThird.text = "0";
    //inventoryThird.text = "0";
  }
  */
  public function getBgContainer():Sprite {
    return backgroundContainer;
  }

  /**
   * Shows or hides the panel, depending on the input string.
   *
   * @param hide: true or false
   */
  public function hideItemsPanel(hide:Boolean):void {
//hide = false;
    backgroundContainer.visible = (!hide);
```

```
    setHideState(hide);
    gameData.setHideScreen(hide);
/*
    if (hide) {//hide items panel
    if (gameData.getDebug()) {
      gameData.print("panel hidden");
    }

    //backgroundContainer.x = 1100;
    backgroundContainer.visible = hide;
    setHideState(hide);
    gameData.setHideScreen(hide);

    }else {//show items panel
    if (gameData.getDebug()) {
      gameData.print("panel visible");
    }
    //backgroundContainer.x = 0;
    backgroundContainer.visible = hide;
    setHideState(hide);
    gameData.setHideScreen(hide);

    }
*/
    changeAll();//refresh items in items panel
    displayInventory();//update inventory of item 1
    //update the amount of remaing points
    currentPts.text = "Remaining Points: " + gameData.getPoints();
  }

  /**
   * Change the buy button, depending on whether or
   * not you can buy that item, on mouse over.
   * @param  e
   */

  public function buy(e:MouseEvent):void {
    //testing(5);
//gameData.setLevel(lvl);
    //buy the item, if the item is the hourglass
    if ((e.currentTarget == itemsArray[0][1]) && (gameData.getLevel() >= 1)) {
    //if the input number of items to buy is a number
    if (!isNaN(Number(quantityToBuyFirst.text))) {
      //trace("asdfasdfs " +quantityToBuyFirst.text);
      var fullcost = (gameData.getItemCost(0) * Number(quantityToBuyFirst.text));
      //if there are enough points to buy the amount of objects needed
      gameData.print (String("cost = " + fullcost));
      if (gameData.getPoints() >= fullcost) {
      gameData.incrementItemCountInInventory(0, Number(quantityToBuyFirst.text));
      //increase the item amount in the inventory
      gameData.setPoints(gameData.getPoints() - fullcost); //subtract the amount
      of points that the items cost
      inventoryFirst.text = String(gameData.getItemCountInInventory(0));//set the
      count in the inventory box
      displayInventory();
```

```
//trace("bought0!");
      }else {
      gameData.print("you don't have that amount of points to spend");
      }
    }else {
      gameData.print("you've input an incorrect amount of items to buy");
    }
    quantityToBuyFirst.text = "1";
    displayInventory();

    //item is not the hourglass
    }else {
//trace("IN!!!!!!!!!!!!!!!!!");
    for (var i:int = 1; i < itemsArray.length; i++) {
      if (e.currentTarget == itemsArray[i][1]) {//find out which buy button has
      been clicked on
      //if item is visible, then the item can potentially be bought
      if (itemsArray[i][0].visible) {
        //if the amount of points is greater than or equal to the cost of the
        item to buy
        if (gameData.getPoints() >= gameData.getItemCost(i)) {
        //if the item has not been bought yet
        if (gameData.getItemCountInInventory(i) <= 0) {
          //current level is correct
  //trace("level num: " + gameData.getLevel() + "\ti+: " + (i + 2));
          if (gameData.getLevel() >= (i + 2)) {
          gameData.incrementItemCountInInventory(i);//buy item
          gameData.setPoints(gameData.getPoints() - gameData.getItemCost(i));
          inventoryFirst.text = String(gameData.getItemCountInInventory(i));//set
          the count in the inventory box
          changeAll();//change item that has been bought
//  trace("boughtOther!");
          }else {//not high enough level
//          trace(">0 level not high enough to buy that item");

          }
        }else {
          gameData.print(">0 item already in inventory");
        }
        }else {
        gameData.print(">0 you don't have that amount of points to spend");
        }
        displayInventory();
        break;
      }
      }
    }
    }
    currentPts.text = "Remaining Points: " + gameData.getPoints();
//testing(6);
//trace("frame: "+itemsArray[i][1].currentFrame);
  }

  /**
   * Takes care of mouse over actions on Buy buttons.
```

```
 * @param  e:  MouseEvent
 */
public function rollOverListener(e:MouseEvent):void {
  //find out which buy button player has clicked on
  //and change its appearance, based on whether item
  //can be bought, has been bought or can be upgraded
  for (var i:int = 0; i < itemsArray.length; i++) {
  //identify the buy button
  if (e.currentTarget == itemsArray[i][1]) {
    if (itemsArray[i][1].currentFrame == 1) {
    itemsArray[i][1].gotoAndPlay(2);
    }else if (itemsArray[i][1].currentFrame == 4) {
    itemsArray[i][1].gotoAndPlay(5);
    }
  }
  //otherwise (if loop hasn't been broken), the item can't be bought - so don't
  do a rollover
  }
}

/**
 * Takes care of mouse out actions on Buy buttons.
 * @param  e:  MouseEvent
 */
public function rollOutListener(e:MouseEvent):void {
  for (var i:int = 0; i < itemsArray.length; i++) {
  if (e.currentTarget == itemsArray[i][1]) {
    if (itemsArray[i][1].currentFrame == 2) {
    itemsArray[i][1].gotoAndPlay(1);
    }else if (itemsArray[i][1].currentFrame == 5) {
    itemsArray[i][1].gotoAndPlay(4);
    }
  }
  }
}

/**
 * Cycles through the buy buttons and changes their current
 * frame, based on the current item being sold.
 */
public function changeAll():void {
  for (var i:int = 0; i < itemsArray.length; i++) {
  //reset buy button
  //itemsArray[i][1].gotoAndPlay(1);
  //if the item is visible and player has reached correct level
  if (itemsArray[i][0].visible) {
    if (i == 0) {//hourglass
    //if enough points and the level is correct
    if ((gameData.getPoints() >= gameData.getItemCost(i))
    && (gameData.getLevel() >= 2)) {
      itemsArray[i][1].gotoAndStop(1);//can buy
      itemsArray[i][1].buttonMode = true;
      itemsArray[i][1].addEventListener(MouseEvent.CLICK, buy);
    }else {
      itemsArray[i][1].gotoAndStop(3);//can't buy
```

```
        itemsArray[i][1].buttonMode = false;
        itemsArray[i][1].removeEventListener(MouseEvent.CLICK, buy);
      }

      //other items (scanners and walls)
      }else {
      //if item not already in inv
      if (gameData.getItemCountInInventory(i) == 0) {
        //if enough points and the level is correct
        if ((gameData.getPoints() >= gameData.getItemCost(i))
        && (gameData.getLevel() >= (i + 2))) {
        if ((i >= 3) && (i <= 5)) {
          itemsArray[i][1].gotoAndStop(4);//can buy
        }else {
          itemsArray[i][1].gotoAndStop(1);//can buy
        }
        itemsArray[i][1].buttonMode = true;
        itemsArray[i][1].addEventListener(MouseEvent.CLICK, buy);
        }else {
        if ((i >= 3) && (i <= 5)) {
          itemsArray[i][1].gotoAndStop(6);//can't buy
        }else {
          itemsArray[i][1].gotoAndStop(3);//can't buy
        }
        itemsArray[i][1].buttonMode = false;
        itemsArray[i][1].removeEventListener(MouseEvent.CLICK, buy);
        }
      }else {//item in inv
        upgradeItem(i);
      }
      }
    }
    }
  }

  /**
   * Changes a specific item, from the items panel,
   * if it can be upgraded, now.
   * @param  indx: index of the current item
   */
  public function upgradeItem(indx:int):void {
    //if the index is 1, 2 or 3, change to indx+2
    if ((indx == 1) || (indx == 2) || (indx == 3)) {
    //hide the current item (since bought)
    itemsArray[indx][0].visible = false;
    //make the next upgrade visible
    itemsArray[indx + 2][0].visible = true;

    //change the buy  button
    //item can be bought
    if ((gameData.getLevel() >= (indx + 2))
    && (gameData.getPoints() >= gameData.getItemCost(indx))
    && (gameData.getItemCountInInventory(indx+2))){
      itemsArray[indx][1].gotoAndPlay(4);
//      trace("done!");
```

```
    }else {//item can't be bought
      itemsArray[indx][1].gotoAndPlay(6);
    }
    //index = 4 or 5
    }else {
    //hide the current item and its buy button(since bought)
    itemsArray[indx][0].visible = false;
    itemsArray[indx][1].visible = false;
    }
  }

  /**
   * Displays the first three items, when the panel is created.
   */
  public function setUpItems():void {
    for (var i:int = 0; i < itemsArray.length; i++) {
    if ((i >= 0) && (i < 3)) {//first three items to be shown
      itemsArray[i][0].visible = true;
    }else {
      itemsArray[i][0].visible = false;
    }

    }

  }

  /**
   * Displays the inventory of the requested object on the correct page (the
   current page).
   */
  public function displayInventory():void {
    //inventory of the first item (hourglass)
    inventoryFirst.text = String(gameData.getItemCountInInventory(0));
  }

  /**
   * Tells whether the panel is hidden or not.
   * @return  Boolean
   */
  public function getHideState():Boolean {
    return hidden;
  }

  /**
   * Sets the state of the panel - whether it's hidden or not.
   * @param  state: Boolean
   */
  public function setHideState(state:Boolean):void {
    hidden = state;
  }

  /**
   * Reset all variables; useful for restarting game.
   */
  private function resetAll():void {
```

```
    backgroundContainer = new Sprite();
    firstItemContainer = new Sprite();
    secondItemContainer = new Sprite();
    thirdItemContainer = new Sprite();
    itemsPanelOutline = new ItemsPanelOutline();
    itemFirst = new ItemFirst();
    buyButtonFirst = new BuyButton();
    itemSecond = new ItemSecond();
    buyButtonSecond = new BuyButton();
    itemThird = new ItemThird();
    buyButtonThird = new BuyButton();
    itemFourth = new ItemFourth();
    itemFifth = new ItemFifth();
    itemSixth = new ItemSixth();
    quantityToBuyFirst = new TextInput();
    inventoryFirst = new TextInput();
    currentPts = new TextField();
    textFormat = new TextFormat();//formatting
    currentPtsTextFormat = new TextFormat();//formatting for the current points
    text
    levelText = new TextField();
    levelTextFormat = new TextFormat();//formatting
    itemsArray = new Array();
    itemsPanelItems = new Array();//all items of this class will be stored here
    upgradesContainer2 = new MouseOverItems();
    upgradesContainer3 = new MouseOverItems();

  }

  }
}
```

Line.as

```
package classes {

  import flash.display.MovieClip;


  public class Line extends MovieClip {
  /**
   * Initializes coords and data for line.
   * @param  X:  x coord
   * @param  Y:  y coord
   * @param  W:   width
   * @param  H:  height
   * @param  R:  rotation
   */
  public function Line(X:int, Y:int, W:int, H:int, R:int) {
    //x and y coords input
    this.x = X;
    this.y = Y;
```

```
    this.scaleX = W;
    this.scaleY = H;
    this.rotation = R;
    this.visible = false;
    this.alpha = 0.5;
  }
  }
}
```

Messages.as

```
package classes {

  import flash.display.MovieClip;
  import flash.display.Sprite;
  import flash.events.MouseEvent;

  import LevelCompleteButton;
  import GameMessages;

  public dynamic class Messages extends MovieClip {
  //contains messages and button
  private static var container:Sprite = new Sprite();
  private static var levelCompleteButton:LevelCompleteButton = new
  LevelCompleteButton();
  private static var gameMessages:GameMessages = new GameMessages();

  public function Messages() {
    gameMessages.x = 194.9;
    gameMessages.y = 94.75;

    levelCompleteButton.x = 344.80;
    levelCompleteButton.y = 402.80;

//    levelCompleteButton.addEventListener(MouseEvent.CLICK, continueGame);


    this.addChild(gameMessages);
    this.addChild(levelCompleteButton);
    this.visible = false;
  }

  /**
   * Advances the current frame of gameMessages.
   *
   * @param  lvl: current level
   */
  public function advanceFrame(lvl:int):void {
//trace("curr frm: " + gameMessages.currentFrame);
    //make sure that the current frame is not the end of the mc
    if (lvl < 8) {
    //increment frame
```

```
    gameMessages.gotoAndStop(lvl);
    }else {
    //restart frame calling
    gameMessages.gotoAndStop(1);
    }
  }

  /**
   * Returns the level complete button to be used to go
   * to the next part of the game.
   *
   * @return LevelCompleteButton
   */
  public function returnButton():LevelCompleteButton {
    return levelCompleteButton;
  }
  }

}
```

Person.as

```
//!need to change the color of the A, after every color change

package classes {

  import flash.display.MovieClip;
  import flash.text.TextField;
  import flash.text.TextFormat;
  import flash.text.TextFieldAutoSize;
  import flash.utils.getTimer;
  import flash.events.TimerEvent;
  import flash.utils.Timer;
  import flash.geom.ColorTransform;

  import classes.Admin;
  /**
   * Creates a Person object with all of its properties.
   * @author Era Vuksani
   */
  public class Person extends MovieClip {


  public var signed:Boolean = false;//person has not signed form yet
  public var collisions:int = 0;//collisions between person and address boxes
  public var color:String = "gray"; //set the color of the current object to be
  green
  public var mood:int = 0;//initially, people are very patient
  public var timer = getTimer();//time of person creation
  public var deltaMood = -1;//how long before color = purple

  public var boxTimer:Number = 0;//box entry time
```

```
  //amount of time that an object can stay in a box, depends on its mood
  public var periodAllowedInBox:Number = -1;
  //if unauthorized, status will change based on these
  public var unauthorizedParam:Array = [null, null]; //[red/not red, leave/stay]
  public var frustratable:Boolean = true; //person can be frustrated
  public var authorized:Boolean = false;//person has unauthorized device until he
  signs the form
  public var moving:Boolean = false; //not moving, since just created
  public var movingToBox:int = -1; //which address space the person is moving to
  public var lineMove:Boolean = true;//person moves down line
  public var formMovement:String = "forward";//move form in that direction
  public var isFading:Boolean = false; //not fading yet
  public var fadeValue:int = -1; //not fading yet
  public var stopFrustration:Boolean = false;//mood stays the same
  private var admin:Boolean = false;//whether or not the person is an admin
//!? scanned?
  private var scanned:Boolean = false;
  public var pauseTime:int = 0;//the amount of time that the object has spent
  being paused

  //steps to move from area2 to area3
  public var stepX:int = -1;
  public var stepY:int = -1;

  public var awesome:int;//for testing purposes

  //A letter for admins
  public var A:TextField = new TextField();
  private var AFormat:TextFormat = new TextFormat();

  //timer for infection spread
  private var infectionTimer:Timer = new Timer(30, 0); //should be 3 sec timer
  private var infectedBy:Array = new Array(0);//location and person that infected
  admin
  //public var admin:Admin = new Admin();
  //private var shieldTimer:Timer = new Timer

  /**
   * Sets up the person object.
   * @param  a:  whether the person can potentially be an admin (ie: lvl >= 4)
   */
  public function Person(a:Boolean){
    this.x = 50;
    this.y = 50;


    if (a) {
    setAdmin(true);//makes the person an admin
    this.mood = 1;
    ///admin = true;
    }

  }

  /**
```

```
 * Adds the specified amount of time to the object's
 * total amount of paused time.
 * @param  t
 */
public function addToPauseTime(t:int):void {
  pauseTime += t;
}

/**
 * Finds out what the fadeValue should be, based on
 * the amount of time that you need to fade the object.
 *
 * @param  time:  time it takes to make obj transparent
 */
public function setFadeValue(time:Number):void {
  fadeValue = 1 / time; //alpha starts at 1
}

/**
 * Start to fade the person (has collided with the edge
 * of the address boxes).
 */
public function render():void {
  if (isFading) {
  alpha -= fadeValue;//fade obj
  //reset alpha to 0, if alpha goes below 0
  if (alpha <= 0) {
    alpha = 0;
    isFading = false;//obj has finished fading
  }
  }
}

/**
 * Figure out when each person is supposed to turn purple.
 *
 * @param  min:  min amount of time to turn blue-gray
 * @param  max:   max amount of time to turn blue-gray
 */
public function timeToPurple(min:int, max:int):void {
  if (deltaMood == -1) {//has not been set before
  //if min equals max, then the time should be min (10sec?)
  if (min == max) {
    deltaMood = min;
  }else{
    deltaMood = Math.floor(Math.random()*(max-min+1))+min;
  }
  }
}

/**
 * Sets the scanned variable, which reflects whether a person
 * has been scanned with a scanner or not.
 * @param sc: whether object has been scanned or not
 */
```

```
public function setScannerVal(sc:Boolean):void {
  scanned = sc;
}

/**
 * Returns if the object has been scanned or not.
 * @return  Boolean
 */
public function getScannerVal():Boolean {
  return scanned;
}

/**
 * Prints out all attributes of this person.
 *
 * @return string of attributes
 */
public function print():String {
  return "awesome: " + this.awesome + "\nsigned: " + this.signed +
  "\ncollisions: " +
  this.collisions + "\ncolor: " + this.color + "\nmood: " +
  this.mood + "\ntimer: " + this.timer + "\nboxTimerAuthorized: " +
  this.boxTimer + "\nperiodAllowedInBox: " + this.periodAllowedInBox +
  "\nunauthorizedParam: " + unauthorizedParam + "\nfrustratable: " +
  this.frustratable +"\nauthorized: " + this.authorized + "\nmoving: " +
  this.moving + "\nmovingToBox: " + this.movingToBox + "\nlineMove: " +
  this.lineMove + "\nformMovement: " + this.formMovement + "\nisFading: " +
  this.isFading +  "\nfadeValue: " + this.fadeValue + "\nstopFrustration: " +
  this.stopFrustration + "/n/n";
}

/**
 * Sets up the text for the A that is added on top
 * of the person object.
 */
public function addAText():void {
  /*
  if (admin){
  //formatting:
  var AFont:Courier = new Courier();
  AFormat.font = AFont.fontName;
  AFormat.color = 0xFFFFFF;
  AFormat.size = 50;

  //set coords of A
  A.x = -17;
  A.y = -30;

  A.defaultTextFormat = AFormat;
  A.text = "A";

  A.selectable = false;
  A.mouseEnabled = false;//mouse will not be able to click on it
  A.autoSize = TextFieldAutoSize.LEFT;
  A.textColor = 0xFFFFFF;
```

```
    //for testing purposes:
    //A.border = false;

    this.addChild(A);//add A to Person
    trace("A color init: " + A.textColor);
    }*/

  }

  /**
   * Resets the admin A's color to white, after the person's
   * color has been changed.
   */
  public function resetAColor():void {
    A.textColor = 0xFFFFFF;
    AFormat.color = 0xFFFFFF;

  }

  /*
  public function changeColor():void {

  }
  */
  /**
   * Returns whether or not the person is an admin.
   * @return  admin: Boolean
   */
  public function getAdmin():Boolean {
    //trace("admin = " + admin);
    return admin;
  }

  /**
   * Sets the value of admin.
   * @param  b:  Boolean
   */
  public function setAdmin(b:Boolean):void {
    admin = b;
    this.gotoAndPlay(2);
  }

  //public function
  /*
  public function makeAdmin():void {
    var High:int = 4;
    var Low:int = 0;
    //1/5 of the people will be admins in the upper levels (4+)
    if ((Math.floor(Math.random() * (1 + High - Low)) + Low) == 0) {
    admin = true;
    addAText();
    trace("admin? " + admin);
    }
  }
```

```
  public function changeColor():void {
    if (!admin) {

    }
  }
  */

  /**
   * Shows a red circle with the Admin logo
   * (for the compromised admin).
   */
  public function compromiseAdmin():void {
    this.gotoAndPlay(3);
    this.color = "red";//compromised color
    this.mood = 4;//red
    //startInfectionTimer();//find out if infection will spread now
  }

  /**
   * Starts the timer that counts 5 sec, and
   * if the timer is finished, Main starts infecting
   * the other people in the subnet.
   */
//!remove
  public function startInfectionTimer():void {
    infectionTimer.start();
    infectionTimer.addEventListener(TimerEvent.TIMER, infectionTimerHandler);
  }

  /**
   * Removes the listener of the timer, once the timer
   * is done.
   * @param  e:  TimerEvent/when timer ends
   * @return  Boolean: timer done or not?
   */
//!remove
  public function infectionTimerHandler(e:TimerEvent):Boolean {
    infectionTimer.removeEventListener(TimerEvent.TIMER, infectionTimerHandler);
    return true;//ie: timer is done
  }

//!remove
//! need to put a pause function for the timer
  public function returnTimer():Timer {
    return infectionTimer;
  }

  /**
   * Returns who the admin was infected by.
   * @return  infectedBy
   */
//!remove
  public function getInfectedBy():Array {
    return infectedBy;
```

```
  }

  /**
   * Sets who the admin was infected by.
   * @param  p:  person
   */
//!remove
  public function setInfectedBy(indx:int, p:Person):void {
    infectedBy.push(indx);//store the index of the compromiser, as well as the
    compromiser
    infectedBy.push(p);
  }

  }

}
```

PointBurst.as

```
package classes{
  import flash.display.*;
  import flash.events.*;
  import flash.text.*;
  import flash.utils.Timer;

  public class PointBurst extends Sprite {
  // text style
  static const fontFace:String = "Arial";
  static const fontSize:int = 15;
  static const fontBold:Boolean = false;
  static const fontColor:Number = 0x000000;

  // animation
  static const animSteps:int = 10;
  static const animStepTime:int = 50;
  static const startScale:Number = 0;
  static const endScale:Number = 2.0;

  private var tField:TextField;
  private var burstSprite:Sprite;
  private var parentMC:MovieClip;
  private var animTimer:Timer;

  public function PointBurst(mc:MovieClip, pts:Object, x,y:Number) {

    // create text format
    var tFormat:TextFormat = new TextFormat();
    tFormat.font = fontFace;
    tFormat.size = fontSize;
    tFormat.bold = fontBold;
    tFormat.color = fontColor;
    tFormat.align = "center";
```

```
    // create text field
    tField = new TextField();
    tField.embedFonts = true;
    tField.selectable = false;
    tField.defaultTextFormat = tFormat;
    tField.autoSize = TextFieldAutoSize.CENTER;
    tField.text = String(pts);
    tField.x = -(tField.width/2);
    tField.y = -(tField.height/2);

    // create sprite
    burstSprite = new Sprite();
    burstSprite.x = x;
    burstSprite.y = y;
    burstSprite.scaleX = startScale;
    burstSprite.scaleY = startScale;
    burstSprite.alpha = 0;
    burstSprite.addChild(tField);
    parentMC = mc;
    parentMC.addChild(burstSprite);

    // start animation
    animTimer = new Timer(animStepTime,animSteps);
    animTimer.addEventListener(TimerEvent.TIMER, rescaleBurst);
    animTimer.addEventListener(TimerEvent.TIMER_COMPLETE, removeBurst);
    animTimer.start();
  }

  // animate
  public function rescaleBurst(event:TimerEvent) {
    // how far along are we
    var percentDone:Number = event.target.currentCount/animSteps;
    // set scale and alpha
    burstSprite.scaleX = (1.0-percentDone)*startScale + percentDone*endScale;
    burstSprite.scaleY = (1.0-percentDone)*startScale + percentDone*endScale;
    burstSprite.alpha = 1.0-percentDone;
  }

  // all done, remove self
  public function removeBurst(event:TimerEvent) {
    burstSprite.removeChild(tField);
    parentMC.removeChild(burstSprite);
    tField = null;
    burstSprite = null;
    delete this;
  }
  }
}
```

SepOutline.as

```
package classes {

  import flash.display.MovieClip;


  public class SepOutline extends MovieClip {

  public function SepOutline(X:int, Y:int, R:int, ... otherArgs){// W:int, H:int,
  R:int) {
    //x and y coords input
    this.x = X;
    this.y = Y;
    this.rotation = R;
    if (otherArgs.length == 1) {
    this.scaleX = otherArgs[0];
    }
    this.visible = false;//not visible, initially.
  }

  /**
   * Changes the height of a separation outline, based on the
   * current level.
   * @param  level
   */
  public function changeHeight(level:int):void {
    if (level > 4) {
    this.scaleX = (1+(1/5));
    //this.scaleY = 1;
    }else {
    this.height = (level / 5) * 60;
    trace("height: " + this.height);
    this.width = 6;
    trace("width: " + this.width);
    }
  }
  }

}
```

Wall.as

```
package classes {

  import flash.display.MovieClip;


  public class Wall extends MovieClip {

  //private var inUse:Boolean = false;//has been bought and is showing
  private var used:Boolean = false;//has been set (ie: player is done with it)
  private var held:Boolean = false;//whether wall is held down by mouse
```

```
//////////////
/**
 * Set the value of inUse.
 * @param  b  boolean (new value of inUse)
 */
/*
public function setInUse(b:Boolean):void {
  inUse = b;
}

/**
 * Returns the value of inUse (if wall is in use or not).
 * @return  Boolean
 */
/*
public function getInUse():Boolean {
  return inUse;
}
//////////////////////
/**
 * Change usedUp, in order to show whether
 * the wall has been used up.
 * @param  b  boolean  (new value of usedUp
 */
public function setUsed(b:Boolean):void {
  used = b;
}

/**
 * Returns the value of usedUp (if wall has been used up or not).
 * @return  Boolean
 */
public function getUsed():Boolean {
  return used;
}

/**
 * Returns whether or not the wall is being held by the mouse
 * or not.
 * @return  Boolean
 */
public function getWallHeld():Boolean {
  //trace("current value of held: " + held);
  return held;
}

/**
 * Set whether the wall is being held by the mouse or not.
 * @param  b:  is wall being held/dragged
 */
public function setWallHeld(b:Boolean):void {
  held = b;
  //trace("new value of held: " + b);
}
}
```

```
}
```

Bibliography

[1] "2012 Data Breach Investigation Report." Verizon. Verizon, 2012. Web. 24 Apr 2012. http://www.verizonbusiness.com/resources/reports/rp_data-breach-investigations-report-2012_en_xg.pdf.

[2] Barab, Sasha A., Brianna Scott, Sinem Siyahhan, Robert Goldstone, Adam Ingram-Goble, Steven J. Zuiker, and Scott Warren. "Transformational Play as a Curricular Scaffold: Using Videogames to Support Science Education." Journal of Science Education and Technology. (2009). Web. 27 Mar. 2012. http://inkido.indiana.edu/research/onlinemanu/papers/barab_jset_2009.pdf.

[3] Dondlinger, Mary Jo. "Educational Video Game Design: A Review of the Literature." Journal of Applied Educational Technology. 4.1 (2007). Web. 27 Mar. 2012. http://www.eduquery.com/jaet/JAET4-1_Dondlinger.pdf.

[4] Gagnon, Diana. "Videogames and Spatial Skills: An Exploratory Study." Educational Communication and Technology. 33.4 (1985): 263-275. Web. 27 Mar. 2012. http://www.jstor.org/stable/30218172.

[5] Gonzlez-Gonzlez, Carina, and Francisco Blanco-Izquierdo. "Designing social videogames for educational uses." Computers & Education: An International Journal. 58. (2012): 250-262. Print.

[6] "Playing Video Games Offers Learning Across Life Span, Say Studies." American Psychological Association. (2008): n. page. Web. 24 Apr. 2012. http://www.apa.org/news/press/releases/2008/08/video-games.asp.

[7] Rosas, Ricardo, Miguel Nussbaum, et al. "Beyond Nintendo: design and assessment of educational video games for first and second grade students." Computers & Education: An International Journal. 40. (2003): 71-94. Print.

[8] Rosenzweig, Gary. Actionscript 3.0 Game Programming University. 2. Indianapolis, IN: Que Publishing, 2011.

[9] Sheng, Steve, Bryant Magnien, Ponnurangam Kumaragugu, Alessandro Acquisti, Lorrie Faith Cranor, Jason Hong, and Elizabeth Nunge. "Anti-Phishing Phil: The Design and Evaluation of a Game That Teaches People Not to Fall for Phish." Web. 27 Mar. 2012. http://cups.cs.cmu.edu/soups/2007/proceedings/p88_sheng.pdf.

[10] Silverman, Barry G., Michael Johns, Ransom Weaver, and Joshua Mosley. "Gameplay, Interactive Drama, and Training: Authoring Edutainment Stories for Online Players (AESOP)." Presence: Teleoperators and Virtual Environments. 16.1 (2007): 65-83. Print.

[11] Terlecki, Melissa S., Nora S. Newcombe, and Michelle Little. "Durable and Generalized Effects of Spatial Experience on Mental Rotation: Gender Differences in Growth Patterns." Applied Cognitive Psychology. 22.996-1013 (2008) Web. 27 Mar. 2012. http://spatiallearning.org/publications_pdfs/terlecki_newcombe_2008.pdf.

[12] Thompson, Michael, and Cynthia Irvine. "Active Learning with the CyberCIEGE Video Game." San Francisco, CA: 2011. Web. 27 Mar. 2012. http://static.usenix.org/event/cset11/tech/final_files/Thompson.pdf.

[13] "Twenty Critical Security Controls for Effective Cyber Defense: Consensus Audit Guidelines (CAG)." SANS. April 15, 2011. Web. 27 Mar 2012. http://www.sans.org/critical-security-controls/cag2.pdf.

[14] "Videogames: Lessons About Learning?." The Teaching Professor. August/September 2004. Print.

[15] Vijayan, Jaikumar. "TJX data breach: At 45.6M card numbers, it's the biggest ever." Computer World. 29 Mar 2007: n. page. Web. 20 Apr. 2012. http://www.computerworld.com/s/article/9014782/TJX_data_breach_At_45.6M_card_numbers_it_s_the_biggest_ever.

[16] Werther, Joseph, Michael Zhivich, Tim Leek, and Nickolai Zeldovich. "Experiences In Cyber Security Education: The MIT Lincoln Laboratory Capture-the-Flag Exercise." Web. 27 Mar 2012. http://people.csail.mit.edu/nickolai/papers/werther-ctf.pdf.

[17] Wikipedia, Flat network. N.p., 2011. Web. 27 Apr 2012. http://en.wikipedia.org/wiki/Flat_network.